PART 3

Network Calculus: Global Analysis

10

Modular Analysis: Computing with Curves

In the previous part, the concepts to compute the performance bounds in one server have been introduced. The aim of this part is to use them to study more complex communication networks. The general problem of computing performance bounds in networks has received much attention, and many techniques have been used, depending on the hypothesis on the arrival and departure processes.

This part is divided into three chapters. The first two chapters focus on the general methods to compute performance bounds in feed-forward networks (i.e. there is no cyclic dependence between the flows circulating in the network). Feed-forward networks are a general class of networks for which the stability condition (i.e. the condition under which the amount of data in the network remains bounded) is known. As a result, the only problem that is focused on here is the problem of computing performance bounds that approach as much as possible the exact worst-case performance bound. The third chapter will be devoted to cyclic networks, where no general result is known for the stability of large classes of networks.

As far as acyclic networks are concerned, there are two main approaches:

1) the first uses the contracts (arrival and service curves) to compute worst-case delay bounds. As we will see, the tightness of the performance bounds we had in the single server case is lost, but the methods developed are very algorithmically efficient. The key tool is the algebraic translation of the serialization of servers as a (min,plus) convolution. This is the object of this chapter;

2) the second directly uses the trajectories of flow satisfying the contracts. As a result, the results are far more precise. This has two consequences: in some cases, it is possible to obtain the exact worst-case performance bounds. This will be the aim of Chapter 11.

Deterministic Network Calculus: From Theory to Practical Implementation,
First Edition. Anne Bouillard, Marc Boyer and Euriell Le Corronc.
© ISTE Ltd 2018. Published by ISTE Ltd and John Wiley & Sons, Inc.

In this chapter, we focus on the performance bounds in networks that can be computed using the (min,plus) framework defined in the previous chapters. We focus on the computation of worst-case delays for a given flow or the worst-case backlog at a server and on acyclic networks.

Several methods can be used, but they rely on the same elements (for the delay computation here):

1) sort the servers in a topological order so that they are analyzed in this order;

2) for each server, compute the residual service curves for each flow and compute the intermediate arrival curves for each flow at the exit of this server;

3) compute a global service curve and an upper bound of the end-to-end delay for each flow.

Step 1 can be easily performed in linear time in the size of the network, by a depth-first-search algorithm, for example (see [COR 09]). Step 2 is carried out using the results of Chapters 5 and 7, and the computation depends on the service policy. Moreover, if a maximum service curve is known, then this step can be improved using *grouping techniques*. Step 3 is carried out using the algebraic properties of the convolution. In this chapter, Step 3 and improvements for Step 2 are detailed.

The chapter is organized as follows: we first present the network model and notations that we will use in this part of the book. Section 10.3 is devoted to the special case of nested tandems, for which better bounds can be obtained using the *pay multiplexing only once phenomenon*. Then, several general algorithms for computing performance bounds in feed-forward networks will be presented in section 10.4, from the naive to the elaborated ones. Finally, in section 10.5, we will prove the NP-hardness of computing exact worst-case delay bounds given the characteristics of the network.

10.1. Network model

10.1.1. *Network description and notations*

In our model, a network $\mathcal{N}$ of n servers and m flows is described by several elements.

Flows. For all i in $\{1, \dots, m\}$, flow i is described by an arrival curve α_i and a path $\mathbf{p}_i = \langle p_i(1), \cdots, p_i(\ell_i)\rangle$ where $p_i(\ell) \in \{1, \dots, n\}$.

We write $h \in \mathbf{p}_i$ or $i \in \mathrm{Fl}(h)$ if there exists $\ell \in \{1, \dots, \ell_i\}$ such that $p_i(\ell) = h$. With a slight abuse of notation, we also interpret $\mathbf{p}_i$ as the set of servers crossed by the flow. If $h = p_i(\ell)$, then $\mathrm{pred}_i(h) = p_i(\ell - 1)$, with the convention that if $\ell = 1$, then $\mathrm{pred}_i(h) = 0$.

We also note $\mathrm{fst}(i) = p_i(1)$ the first server visited by i, and $\mathrm{end}(i)$ the last system visited by i, i.e. $p_i(\ell_i)$. Bits of data of flow i successively cross each server of $\mathbf{p}_i$ in the order given by the path.

The cumulative arrival process of flow i before its visited server is denoted by $F_i^{(0)}$ and is α_i-constrained. For $h \in \mathbf{p}_i$, we denote by $F_i^{(h)}$ the cumulative departure process of flow i after server h.

Servers. For all h in $\{1, \ldots, n\}$, server $\mathcal{S}^{(h)}$ – or simply server h – is described by a service curve $\beta^{(h)} \in \mathcal{C}$, a type of service $\mathcal{T}^{(h)}$ (strict or min-plus) and a service policy (this can be arbitrary, FIFO, static priorities, general processor sharing, for example). For each server $h \in \{1, \ldots, n\}$, the aggregate server is $\mathcal{S}^{(h)} = \mathcal{S}_{\mathcal{T}^{(h)}}(\beta^{(h)})$.

With the notations introduced above, we have that for all $h \in \{1, \ldots, n\}$,

$$((F_i^{(\mathrm{pred}_i(h))})_{i \in \mathrm{Fl}(h)}, (F_i^{(h)})_{i \in \mathrm{Fl}(h)}) \in \mathcal{S}^{(h)}.$$

For server h and $t \geqslant 0$, we denote by $\mathit{Start}_h(t)$ the start of the backlogged period of server h at time t.

The graph induced by this network is the directed graph $G_{\mathcal{N}} = (V, E)$, where $V = \{1, \ldots, n\}$ is the set of the servers and $E = \{(h, \ell) \in V \times V \mid \exists i \in \{1, \ldots, m\},\ \mathbf{p}_i = \langle \ldots, h, \ell, \ldots \rangle\}$ is the pair of consecutive servers on a flow. As a result, $G_{\mathcal{N}}$ is a directed and simple graph (with no loop and no multiple edge).

10.2. Some special topologies

In this part, we will focus on some topologies illustrated in Figure 10.1.

Feed-forward networks. A feed-forward network is a network $\mathcal{N}$ whose induced graph $G_{\mathcal{N}}$ is acyclic. In that case, it is possible to number the servers so that the path of each flow is increasing: $\forall i \in \{1, \ldots, m\}, \forall \ell < \ell_i, p_i(\ell) < p_i(\ell + 1)$. This numbering can be obtained by performing a topological sort (see, for example, [COR 09]). We will always assume this type of numbering when dealing with acyclic networks (in Chapters 10 and 11).

Tandem networks. A tandem network is a special case of feed-forward network with a line topology: the edges of the induced network are $(h, h+1)$, $h \in \{1, \ldots, n-1\}$. As a result, the path of each flow is of the form $\langle \mathrm{fst}(i), \mathrm{fst}(i)+1, \ldots, \mathrm{end}(i) \rangle$.

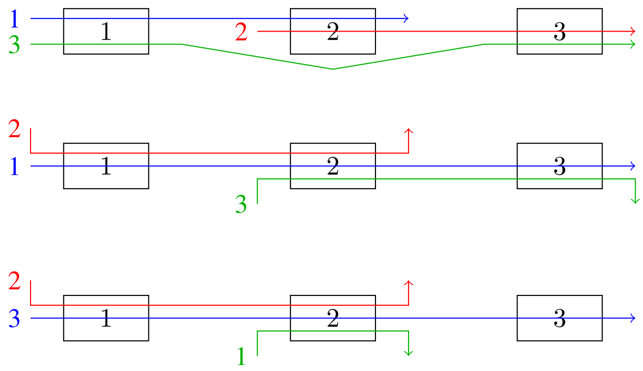


Figure 10.1. *Examples of topologies of networks. Top: a feed-forward network but not tandem; middle: a tandem network but not nested; bottom a nested tandem network. For a color version of this figure, see www.iste.co.uk/bouillard/calculus.zip*

Nested tandem networks. Finally, a nested tandem network is a tandem network such that the path of each flow is included in the path of the other. Then, the flows can be numbered so that

$$(Nest) \qquad \forall i, j \in \{1, \dots, m\}, \quad i < j \Leftrightarrow \mathbf{p}_i \subseteq \mathbf{p}_j.$$

10.3. The pay multiplexing only once phenomenon

Consider the network of Figure 10.2, with two flows crossing two servers.

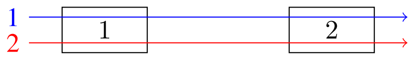


Figure 10.2. *Network with two flows and two servers to illustrate the pay multiplexing only once phenomenon. For a color version of this figure, see www.iste.co.uk/bouillard/calculus.zip*

Suppose that in the first server, a burst of data of flow 1 is served before the data of flow 2, although data of flow 2 arrived before the burst in the server. Then, in the second server, this burst of data of flow 1 cannot overtake the data of flow 2 again, as it already arrived before in the second server. However, when we apply Theorems 6.1 and 7.1, we obtain the residual min-plus service curve

$$\tilde{\beta}_1 = \left[\beta^{(1)} - \alpha_1\right]_{\uparrow}^{+} * \left[\beta^{(2)} - \alpha_1^{(1)}\right]_{\uparrow}^{+},$$

where $\alpha^{(1)} = \alpha_1 \oslash \left[\beta^{(1)} - \alpha_2\right]_{\uparrow}^{+}$ is the arrival curve flow 1 at server 2. In this formula, the *multiplexing* of the two flows appears in the two servers, meaning that a bit of data can suffer from the same burst of flow 1 in each server, which is too pessimistic. In some cases, however, direct computation gives better results.

Indeed, suppose that the service curves are strict and that the service policy is blind. Let $t \in \mathbb{R}_+$. We can define $s = \mathit{Start}_2(t)$ and $u = \mathit{Start}_1(s)$, the respective start of backlog period of t at server 2 and of s at server 1. We can write:

$$F_1^{(2)}(t) + F_2^{(2)}(t) \geqslant F_1^{(1)}(s) + F_2^{(1)}(s) + \beta^{(2)}(t-s) \quad \text{and}$$
$$F_1^{(1)}(s) + F_2^{(1)}(s) \geqslant F_1^{(0)}(u) + F_2^{(0)}(u) + \beta^{(1)}(s-u),$$

inducing

$$F_1^{(2)}(t) + F_2^{(2)}(t) \geqslant F_1^{(0)}(u) + F_2^{(0)}(u) + \beta^{(1)}(s-u) + \beta^{(2)}(t-s).$$

As $F_1^{(2)}(t) \leqslant F_1^{(0)}(t)$ and $F_2^{(2)}(t) \geqslant F_2^{(1)}(s) \geqslant F_1^{(0)}(u)$, we obtain

$$F_2^{(2)}(t) \geqslant F_1^{(0)}(u) + (\beta^{(1)} * \beta^{(2)} - \alpha_1)^{+}(t-u) \qquad [10.1]$$

and $\tilde{\beta}_2 = (\beta^{(1)} * \beta^{(2)} - \alpha_1)^{+}$ is a service curve for flow 2. We call this the *pay multiplexing only once (PMOO)* phenomenon.

This concept was introduced by Schmitt *et al.* in [SCH 08a, SCH 08b, SCH 06b, SCH 07], and it can be noted that this result cannot be found by applying Theorem 6.1 and then Theorem 7.1 as the service curve obtained by Theorem 6.1 is not strict. However, this example shows that direct computation gives the same result.

10.3.1. *Loss of tightness*

The PMOO phenomenon could be generalized to an arbitrary network, but this is not the solution to always compute better performance bounds, although this intuitively should be the case as the burst of flow 1 is counted only once. In this section, we illustrate the difficulty of getting good bounds.

We still consider the small network of Figure 10.2 with the following parameters:

– for $h \in \{1, 2\}$, server h offers a strict service curve $\beta^{(h)} = \beta_{R_h,T_h}$, and for each server, the service is arbitrary;

– for $i \in \{1, 2\}$, flow i is constrained by the arrival curve $\alpha_i = \gamma_{r_i,b_i}$,

and compare the two service curves obtained, $\tilde{\beta}_1$ and $\tilde{\beta}_2$.

$$\tilde{\beta}_1(t) = (\min(R_1, R_2) - r_1)\left[t - \frac{b_1 + R_1T_1}{R_1 - r_1} - \frac{b_1 + r_1T_1 + R_2T_2}{R_2 - r_1}\right]^+$$

and

$$\tilde{\beta}_2(t) = (\min(R_1, R_2) - r_1)\left[t - \frac{b_1 + \min(R_1, R_2)(T_1 + T_2)}{\min(R_1, R_2) - r_1}\right]^+.$$

If $\beta_1 = \beta_2$, then $\tilde{\beta}_2 \leqslant \tilde{\beta}_1$. But if $b_1 = 0$, $T_1 = 0$ and $R_2 > R_1$, then $\tilde{\beta}_1 \leqslant \tilde{\beta}_2$.

The first method, used to compute $\tilde{\beta}_1$, can be generalized for general feed-forward networks. This is the object of section 10.4. The second method, leading to $\tilde{\beta}_2$, cannot be generalized in such a simple way when more flows interfere, and especially when the flows are not *nested*. The following section shows that the formula can be generalized for tandem networks under arbitrary multiplexing. This result might be difficult to apply to specific service policies. One reason, illustrated in section 10.3.3, is that service policies are not stable under composition of servers. However, in section 10.3.4, we show how to benefit from the PMOO phenomenon in nested tandem for static priority and FIFO policies. Another way to generalize this approach is to use linear programming. Indeed, it follows more or less the same computing scheme: first work with the trajectories and bound, using the arrival and service curves only in the last step. This will be discussed in Chapter 11.

10.3.2. *A multi-dimensional operator for PMOO*

In this section, we generalize the PMOO principle to any tandem network. This result is taken from [BOU 08a].

THEOREM 10.1 (PMOO multi-dimensional operator).– *Consider a tandem network $\mathcal{N}$ where flow 1 crosses all of the servers. If each server h offers a strict service* *curve $\beta^{(h)}$, with the same assumptions and notation as above, a min-plus service curve offered for flow 1 is:*

$$\beta(t) = \left[\inf_{\substack{u_1, \dots, u_n \geqslant 0 \\ u_1 + \cdots + u_n = t}} \sum_{j=1}^{n} \beta^{(j)}(u_j) - \sum_{i=2}^{k} \alpha_i(\sum_{j=\mathrm{fst}(i)}^{\mathrm{end}(i)} u_j) \right]^+ .$$

We first prove a lemma that concerns the cumulative processes.

LEMMA 10.1.– *With the previous assumptions and notation,* $\forall t \in \mathbb{R}_+$, $\exists u_1, \dots,$ $u_n \in \mathbb{R}_+$ *such that*

$$F_1^{(n)}(t) - F_1^{(0)}(t - \sum_{j=1}^{n} u_j) \geqslant \sum_{j=1}^{n} \beta_j(u_j)$$

$$- \sum_{i=2}^{k} \left(F_i^{(\mathrm{end}(i))}(t - \sum_{j=\mathrm{end}(i)+1}^{n} u_j) - F_i^{(0)}(t - \sum_{j=\mathrm{fst}(i)}^{n} u_j)\right)$$

Moreover, $F_1^{(n)}(t) - F_1^{(0)}(t - \sum_{j=1}^{n} u_j) \geqslant 0$.

PROOF.– To ease the notations, we will identify $F_i^{(0)}$ with $F_i^{(\mathrm{fst}_i - 1)}$.

Set $s_{n+1} = t$ and for all $h \in \{1, \dots, n\}$, define $s_h = \mathit{Start}_h(s_{h+1})$ and $u_h = s_{h+1} - s_h$.

Since the servers offer strict service curves, we have that for all servers h,

$$\sum_{i \in \mathrm{Fl}(h)} F_i^{(h)}(s_{h+1}) \geqslant \beta^{(h)}(s_{h+1} - s_h) + \sum_{i \in \mathrm{Fl}(h)} F_i^{(h-1)}(s_h). \qquad [10.2]$$

Summing all those inequalities and collecting terms with regard to flow indices, we get

$$\sum_{i=1}^{m} \sum_{h=\mathrm{fst}(i)}^{\mathrm{end}(i)} F_i^{(h)}(s_{h+1}) \geqslant \sum_{h=1}^{n} \beta^{(h)}(u_h) + \sum_{i=1}^{m} \sum_{h=\mathrm{fst}(i)}^{\mathrm{end}(i)} F_i^{(h-1)}(s_h).$$

By removing the terms present on both sides, it simplifies into:

$$\sum_{i=1}^{m} F_i^{(\mathrm{end}(i))}(s_{\mathrm{end}(i)+1}) \geqslant \sum_{h=1}^{n} \beta^{(h)}(u_h) + \sum_{i=1}^{m} F_i^{(\mathrm{fst}(i)-1)}(s_{\mathrm{fst}(i)}).$$

which gives the first inequality of the lemma, since $s_h = t - \sum_{j=h}^{n} u_j$.

Moreover, we have for all flow i, $F_i^{(h)}(s_{h+1}) \geqslant F_i^{(h)}(s_h) = F_i^{(h-1)}(s_h)$, so $F_i^{(\mathrm{end}(i))}(s_{\mathrm{end}(i)+1}) \geqslant F_i^{(\mathrm{fst}(i)-1)}(s_{\mathrm{fst}(i)})$, which gives the second inequality. □

We can deduce from this lemma a residual service curve provided to flow 1 for its whole path.

PROOF OF THEOREM 10.1.– Consider the inequalities proved in Lemma 10.1. We clearly have $\forall i \in \{1\dots,m\}$, $\forall h \in \{\mathrm{fst}(i),\dots,\mathrm{end}(i)\}$, $\forall t \in \mathbb{R}_+$, $F_i^{(h)}(t) \leqslant F_i^{(0)}(t)$. Moreover, for all $2 \leqslant i \leqslant m$,

$$\begin{aligned} F_i^{(\mathrm{end}(i))}(s_{\mathrm{end}(i)+1}) - F_i^{(0)}(s_{\mathrm{fst}(i)}) &\leqslant F_i^{(0)}(s_{\mathrm{end}(i)+1}) - F_i^{(0)}(s_{\mathrm{fst}(i)}) \\ &\leqslant \alpha_i(\sum_{j=\mathrm{fst}(i)}^{\mathrm{end}(i)} u_j). \end{aligned}$$

Thus,

$$F_1^{(n)}(t) - F_1^{(0)}(t - \sum_{j=1}^{n} u_j) \geqslant \sum_{h=1}^{n} \beta^{(h)}(u_h) - \sum_{i=1}^{k} \alpha_i(\sum_{j=\mathrm{fst}(i)}^{\mathrm{end}(i)} u_j).$$

From Lemma 10.1, we also have $F_1^{(n)}(t) - F_1^{(0)}(t - \sum_{j=1}^{n} u_j) \geqslant 0$. As a result of those two inequalities, we have $F_1^{(n)} \geqslant F_1^{(0)} * \beta$, where β is the function defined above. □

EXAMPLE 10.1.– *To illustrate the PMOO phenomenon, consider the network of Figure 10.1 (middle), which is the example detailed by Schmitt* et al. *in [SCH 06b, section 4]. Theorem 10.1 provides the following service curve for the path:*

$$\begin{aligned} \beta(t) = \Big[\inf_{\substack{u_1, u_2, u_3 \geqslant 0 \\ u_1 + u_2 + u_3 = t}} & \beta^{(1)}(u_1) + \beta^{(2)}(u_2) + \beta^{(3)}(u_3) \\ & - \alpha_1(u_1 + u_2) - \alpha_2(u_2 + u_3)\Big]^+ . \end{aligned}$$

This result introduces a multi-dimensional operator for network calculus. It generalizes Theorem 7.1, which tackles paths with a single node and a unique cross-traffic flow. When service curves are rate-latency ($\beta^{(h)} = \beta_{R_h,T_h}$) and the arrival curves are leaky-buckets ($\alpha_i = \gamma_{r_i,b_i}$), the formula becomes:

$$\beta(t) = \Big[\inf_{\substack{u_1,\dots,u_n \geqslant 0 \\ u_1 + \cdots + u_n = t}} \sum_{h=1}^{n} \beta^{(h)}(u_h) - \sum_{i=1}^{m} \alpha_i(\sum_{j=\mathrm{fst}(i)}^{\mathrm{end}(i)} u_j)\Big]^+$$

$$= \Big[-\sum_{i=1}^{n} b_i + \inf_{\substack{u_1,\dots,u_n \geqslant 0 \\ u_1+\cdots+u_n = t}} \sum_{h=1}^{n} \big(R_h[u_h - T_h]^+ - (\sum_{i\in \mathrm{Fl}(h)} r_i)u_h \big) \Big]^+$$

$$= \Big[-\sum_{i=1}^{n} b_i + (\gamma_1 * \cdots * \gamma_n)(t) \Big]^+,$$

with $\gamma_h(t) = R_h[t - T_h]^+ - (\sum_{i\in \mathrm{Fl}(h)} r_i)t$, which is a convex function composed of two segments: a segment of slope $-\sum_{i\in \mathrm{Fl}(h)} r_i$ between 0 and T_j and an infinite segment of slope $R_h - \sum_{i\in \mathrm{Fl}(h)} r_i$. Therefore, $(\gamma_1 * \cdots * \gamma_n)$ can be easily computed using Theorem 4.1. The finite segments have negative slopes, and we are interested only in the part where the convolution is greater than $\sum_{i=1}^{m} b_i$, which is of slope $R = \min_{h=1}^{n}(R_h - \sum_{i\in \mathrm{Fl}(h)} r_i)$. After computation, we obtain that $\beta = \beta_{R,T}$ with

$$R = \min_{h=1}^{n}(R_h - \sum_{i\in \mathrm{Fl}(h)} r_i) \quad \text{and} \quad T = \sum_{h=1}^{n} T_h \big(1 + \frac{\sum_{i\in \mathrm{Fl}(h)} r_i}{R}\big) + \sum_{i=1}^{m} \frac{b_i}{R}.$$

10.3.3. *Composition and service policies*

WARNING.– The PMOO phenomenon must be used with care. Indeed, the previous theorem does not imply that a system composed of two servers with the same service policy obeys this service policy. It is true in a few special cases: blind multiplexing (this is obvious as there is no assumption for the service policy) and FIFO multiplexing. Indeed, if a bit of data a arrives before a bit of data b, then a will depart from the first server before b, and the same will hold for server 2.

However, this is not true for other service policies. Let us illustrate this fact with two service policies: static priority (SP) and generalized processor sharing (GPS). In the following example, we consider the simple network in Figure 10.2.

– *Static priority*: this can be deduced from the fact that the end-to-end service curve of two servers in tandem is not a strict service curve. Consider Figure 6.5 and suppose that the flow drawn is the one that is given the highest priority. Consider now a second flow (with lower priority). As the service curves are pure delays, if data from the second flow arrive at time 0, those data will exit at time $T_1 + T_2$. But at that time, flow 1 is still backlogged, which means that the system composed of the two servers is globally not a static priority server.

– *Generalized processor sharing*: take $\beta^{(1)} = \lambda_4$, $\beta^{(2)} = \lambda_5$, $\alpha_1 = \alpha_2 = \gamma_{1,10}$ and $\phi_1 = \phi_2$ the proportion of service guaranteed for flows 1 and 2 for each server. Suppose that the first flow arrives according to α_1 and that half of the burst is served at once, at time 0^+. Flow 2 arrives from time 1, according to $\alpha_2(t-1)$. Between times 0 and 1, only flow 1 is served. From time 0^+, the servers serve data exactly according to their arrival curve. From time 1, server 1 is backlogged for both flows. Then, each flow arrives at rate 2 in server 2.

In server 2, flow 2 cannot be backlogged (it is guaranteed service rate 2.5), so is served at rate 2. Then, flow 1, which is backlogged, is served at rate $5 - 2 = 3$. As a result, flow 2 is not guaranteed half of the service of the global system. One can invert the role of flows 1 and 2 and then can conclude that neither of the two flows can be guaranteed half of the service of the global system. Hence, the global system is not GPS.

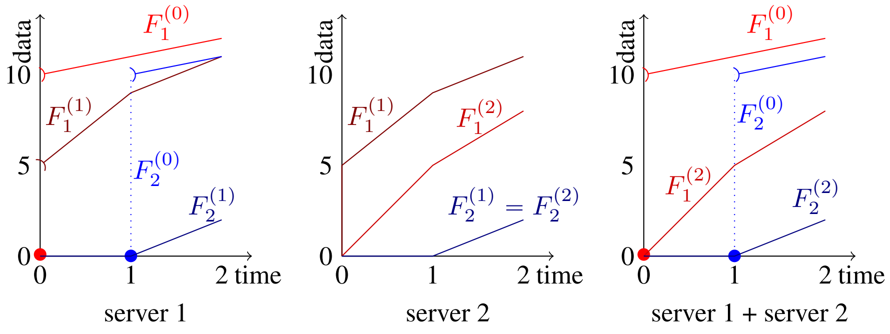


Figure 10.3. *Two GPS servers in tandem is not a GPS server. For a color version of this figure, see www.iste.co.uk/bouillard/calculus.zip*

10.3.4. *Nested tandems*

Nested tandem systems are a special case where PMOO can be generalized for several service policies. We already gave a general formula for blind multiplexing (Theorem 10.1), and in this section, we show that this phenomenon can be taken into account for fixed priorities and FIFO service policies. Intuitively, this is possible when flows can be *removed* one by one, from the nested to the longest. Two cases are explored: the nested flows are given a higher priority (equation [10.1] can be directly applied) or the service policy is FIFO, because the service policy is preserved by the composition of servers.

10.3.4.1. *Static priorities and PMOO*

Suppose that a nested tandem respecting the numbering $(Nest)$ is crossed by m flows and that static priority is at work, respecting the priority rule that flows with the highest number have the lowest priority. In other words, if $\mathbf{p}_i \subseteq \mathbf{p}_j$, then flow i has higher priority than flow j. The service curves $\tilde{\beta}_i$ for each flow i can be computed using Algorithm 3. This is a straightforward generalization of equation [10.1], and the proof is similar to that of Theorem 10.1, so we do not give details here. A more rigorous proof can be found in [BOU 09].

Algorithm 3: PMOO for static priority service policy.

```
Data: $(\alpha_i)_{i=1}^m \in \mathcal{F}^m$, $(\beta^{(h)})_{h=1}^n \in \mathcal{F}^n$, $(\mathbf{p}_i)_{i=1}^m$ increasing and nested paths.
Result: $\tilde{\beta}_i$, the service curve for flow $i$
1 begin
2     for $i = 1$ to $n$ do
3         $PF(i) \leftarrow \{j < i \mid \mathbf{p}_j \subseteq \mathbf{p}_i$ and $\nexists k,\ i > k > j,\ \mathbf{p}_j \subseteq \mathbf{p}_k\}$;
4         $N(i) \leftarrow \mathbf{p}_i \backslash \left(\bigcup_{j<i} \mathbf{p}_j\right)$;
5         $\tilde{\beta}_i \leftarrow \ast_{h \in N(i)} \beta^{(h)} * \ast_{j \in PF(i)} \left[\tilde{\beta}_j - \alpha_j\right]_{\uparrow}^{+}$.
```

EXAMPLE 10.2.– *Consider the nested network at the bottom of Figure 10.1. We compute*

$$\begin{cases} \tilde{\beta}_1 = \beta^{(2)}, \\ \tilde{\beta}_2 = \beta^{(1)} * \left[\tilde{\beta}_1 - \alpha_1\right]_{\uparrow}^{+} = \beta^{(1)} * \left[\beta^{(2)} - \alpha_1\right]_{\uparrow}^{+} \\ \tilde{\beta}_3 = \left[\tilde{\beta}_2 - \alpha_2\right]_{\uparrow}^{+} * \beta^{(3)} = \left[\beta^{(1)} * \left[\beta^{(2)} - \alpha_1\right]_{\uparrow}^{+} - \alpha_2\right]_{\uparrow}^{+} * \beta^{(3)}. \end{cases}$$

10.3.4.2. *FIFO and PMOO*

The same can be done for FIFO multiplexing using the residual service curves thanks to the following three facts:

1) only min-plus service curves are considered;

2) the concatenation of FIFO servers in tandem results in an FIFO system;

3) when considering only a subset of flows, the service policy among those flows only is also FIFO.

The following proposition is a direct consequence of Theorems 7.5 and 6.1, and is used recursively in Algorithm 4. The principle of this algorithm is the same as above: we remove the flows one by one, beginning with the most nested. For each flow, a new residual service curve is computed for the other flows. We use the following proposition.

PROPOSITION 10.1.– *Consider a sequence of n FIFO servers offering respective min-plus service curves $\beta^{(h)}$ crossed by m flows. Suppose that flow 1 is constrained by the arrival curve α. Then, for all $\theta > 0$,*

$$\left[\mathop{\ast}_{h=1}^{n} \beta^{(h)} - \alpha * \delta_\theta\right] \wedge \delta_\theta$$

is a min-plus service curve for the $m - 1$ other flows.

Algorithm 4: PMOO for FIFO service policy in nested tandems.

```
Data: $(\alpha_i)_{i=1}^m \in \mathcal{F}^m$, $(\beta^{(h)})_{h=1}^n \in \mathcal{F}^n$, $(\mathbf{p}_i)_{i=1}^m$ increasing and nested paths,
      $(\theta_i)_{i=1}^m$, parameters for the FIFO residual service curves.
Result: $\tilde{\beta}_n$, the residual service curve for a flow crossing the tandem network
        with flows 1,..., $m$ as cross-traffic.
1 begin
2     for $i = 1$ to $n$ do
3         $PF(i) \leftarrow \{j < i \mid \mathbf{p}_j \subseteq \mathbf{p}_i \text{ and } \nexists k,\ i > k > j,\ \mathbf{p}_j \subseteq \mathbf{p}_k\}$;
4         $N(i) \leftarrow \mathbf{p}_i \backslash \left(\bigcap_{j<i} \mathbf{p}_j\right)$;
5         $\tilde{\beta}_i \leftarrow \left[\left(\ast_{h \in N(i)} \beta^{(h)} * \ast_{j \in PF(i)} \tilde{\beta}_j\right) - \alpha_i * \delta_{\theta_i}\right]^+ \wedge \delta_{\theta_i}$.
```

A major drawback of this algorithm is that the parameters θ_i must be fixed in advance. If analytic formulas are obtained, then it is also possible to perform an optimization step by computing the values of (θ_i) which, for example, will give the least delay bound. This is in general a very difficult problem that has been extensively studied by Lenzini, Stea and others. In [LEN 06], it is proved that in the case of *sink-tree* tandem topologies (all flows stop at the last server), the optimal parameters can be computed recursively, and these parameters lead to a *tight* residual service curve, in the sense that a tight delay bound can be computed from the residual service curve. In [LEN 08], the approach is extended to any tandem topology by *cutting* flows in order to obtain a nested tandem. Finally, in [BIS 08], a simple example shows evidence that the tightness is obtained only for sink-tree topologies.

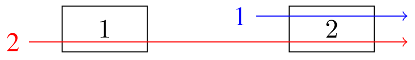


Figure 10.4. *Sink-tree tandem network. For a color version of this figure, see www.iste.co.uk/bouillard/calculus.zip*

EXAMPLE 10.3.– *As an example of the optimization of parameters θ_1, inspired by [LEN 06, Theorem 4.6], let us consider the very simple example of Figure 10.4, and let us compute the residual service curve that would be obtained for a flow crossing the two servers. We assume that arrival curve for flow i is γ_{r_i,b_i} constrained and that server h offers the min-plus service curve β_{R_h,T_h}. We first compute*

$$\tilde{\beta}_1 = [\beta^{(2)} - \alpha_1 * \delta_{\theta_1}]^+ \wedge \delta_{\theta_1}.$$

From Example 7.1, we can consider only values of θ_1 *that are greater than* $C_1 = T_2 + \frac{b_1}{R_2}$*. The residual curve obtained is*

$$\tilde{\beta}_1 = \delta_{\theta_1} * \gamma_{R_2 - r_1, R_2(\theta_1 - C_1)}.$$

As a second step of the algorithm, $\tilde{\beta}_2 = \left[(\beta^{(1)} * \tilde{\beta}_1) - \alpha_2 * \delta_{\theta_2}\right]^+ \wedge \delta_{\theta_2}$*. First,* $\beta^{(1)} * \tilde{\beta}_1 = \delta_{\theta_1 + T_1} * (\gamma_{R_2 - r_1, R_2(\theta_1 - C_1)} \wedge \gamma_{R_1, 0})$*. Here again, we can discard values of* θ_2 *that do not satisfy* $\alpha_2 * \delta_{\theta_2}(\theta_2+) \leqslant \beta^{(1)} * \tilde{\beta}_1(\theta_2)$*, as they will not be optimal. Then,* θ_2 *is lower bounded by*

$$C_2 = \theta_1 + T_1 + \max\left(\frac{b_2}{R_1}, \frac{b_2}{R_2 - r_1} - \frac{(\theta_1 - C_1)R_2}{R_2 - r_1}\right).$$

Then, $\tilde{\beta}_2$ *is of the form* $\delta_{\theta_2} * (\gamma_{r'_1, b'_1} \wedge \gamma_{r'_2, b'_2})$ *with*

$$\begin{aligned} r'_1 &= R_1 - r_2 & b'_1 &= R_1(\theta_2 - \theta_1 - T_1) - b_2 \\ r'_2 &= R_2 - r_1 - r_2 & b'_2 &= (R_2 - r_1)(\theta_2 - \theta_1 - T_1) + (\theta_1 - C_1)R_2 - b_2. \end{aligned}$$

Now, suppose that a flow constrained by the arrival curve γ_{r_3, b_3} *crosses this tandem network. The residual service curve for flow 3 is then* $\tilde{\beta}_2$*. Under stability conditions (*$R_1 > r_2 + r_3$ *and* $R_2 > r_1 + r_2 + r_3$*), the worst-case delay is found as* $\tilde{\beta}_2^{-1}(b_3)$*, i.e.*

$$d = \theta_2 + \max\left(\frac{[b_3 - b'_1]^+}{r'_1}, \frac{[b_3 - b'_2]^+}{r'_2}\right).$$

This delay depends on both θ_1 *and* θ_2*, but we can first fix* θ_1 *and compute the optimal value of* θ_2 *for this fixed* θ_1*, and finally optimize* θ_1*.*

After computations, we find that the optimal value for θ_2 *is* $\bar{\theta}_2 = \theta_1 + T_1 + \max\left(\frac{b_2 + b_3}{R_1}, \frac{b_2 + b_3 - (\theta_1 - C_1)R_2}{R_2 - r_1}\right)$*. In that case,* $d = \theta_2 = \bar{\theta}_2$*. Thus, it is easy to see that to minimize the maximal delay,* θ_1 *has to be minimized, and* $\theta_1 = C_1 = T_2 + \frac{b_1}{R_2}$*. Finally, we find:*

$$d = T_1 + T_2 + \frac{b_1}{R_2} + \max\left(\frac{b_2 + b_3}{R_1}, \frac{b_2 + b_3}{R_2 - r_1}\right).$$

10.4. Per-flow analysis of networks

In this section, we first give several ways to compute worst-case delay upper bounds, from the most naive method to more efficient ones.

10.4.1. *Total output analysis*

If only the results of Chapter 5 are used, a delay bound can be computed using Algorithm 5. It requests that the service curves are strict.

Algorithm 5: Generic TOA (Total output analysis)

```
Data: Acyclic network description : $\mathbf{p}_i$, $\alpha_i$, $\beta^{(h)}$ strict service curves.
Result: Worst-case delay upper bound for each server
1 begin
2     for $h = 1$ to $n$ do
3         $\alpha^{(h)} \leftarrow \sum_{(\ell,h)\in E} \alpha^{(\ell)} \oslash \beta^{(\ell)} + \sum_{i \mid \mathrm{fst}(i)=\ell} \alpha_i$;
4         $d^{(h)} \leftarrow \inf\{t > 0 \mid \alpha^{(h)}(t) \leqslant \beta^{(h)}(t)\}$
5     for $i = 1$ to $m$ do
6         $d_i \leftarrow \sum_{h\in\mathbf{p}_i} d^{(h)}$.
```

In line 3, $\alpha^{(h)}$ is an arrival curve for the arrival cumulative function at server h. Once a flow is aggregated, there is no operation to compute curves for individual flows, hence the pessimism. Then, $d^{(h)}$ is the maximum delay suffered by a bit of data in a server offering a strict service curve $\beta^{(h)}$ and with arrival curve $\alpha^{(h)}$. As the flow is an aggregation of several flows with no *a priori* service policy, this flow cannot be considered as FIFO, and the delay cannot be computed as the horizontal distance between the arrival and service curves. Instead, the maximum length of a backlogged period has to be considered, which corresponds to the formula of line 4 (Theorem 5.5).

This algorithm is very pessimistic as it does not take into account the service policies. Nevertheless, it can be used under very general assumptions: if FIFO-per-flow is not required, which is not possible for the other methods presented hereafter (note that when the flows are not FIFO, better methods exist; see, for example, [SCH 11]).

EXAMPLE 10.4.– *Consider the non-tandem network in Figure 10.1 (top). We find*

$$\begin{cases} \alpha^{(1)} = \alpha_1 + \alpha_3 \\ \alpha^{(2)} = \alpha_2 + (\alpha_1 + \alpha_3) \oslash \beta^{(1)} \\ \alpha^{(3)} = (\alpha_1 + \alpha_3) \oslash \beta^{(1)} + (\alpha_2 + (\alpha_1 + \alpha_3) \oslash \beta^{(1)}) \oslash \beta^{(2)}. \end{cases}$$

and the delays can be directly derived from these formulas.

Explained this way, the method is very pessimistic, especially because it computes performances as if all of the data served by a given server is transmitted to all of its successors. It is possible to circumvent this by using Theorem 5.4 and replacing line 3 with

$$\alpha^{(h)} \leftarrow \sum_{i \in \mathrm{Fl}(\ell)} \alpha_i + \sum_{(\ell,h)\in E} \alpha^{(\ell)} \oslash \beta^{(\ell)}(0),$$

or

$$\alpha^{(h)} \leftarrow \sum_{i \in \mathrm{Fl}(\ell)} \alpha_i + \left(\sum_{(\ell,h)\in E} \alpha^{(\ell)} \oslash \beta^{(\ell)}(0) - \sum_{i \in \mathrm{Fl}(h), fs(i)\neq \ell} \alpha_i(0+) \right) \wedge \delta_0,$$

depending on the shape of the arrival curves. A similar technique is used in [BON 16b].

It should be obvious that this method can be improved. This is done in several ways:

1) taking into account the FIFO-per-flow assumption and the service policy. For example, if the servers are FIFO, then line 4 can be replaced by $d^{(h)} = $ *hDev*$(\alpha^{(h)}, \beta^{(h)})$ and min-plus service curves can be considered;

2) taking into account the pay burst only once phenomenon, corresponding to the composition of servers, which is the object of the rest of the section;

3) taking into account the pay multiplexing only once phenomenon when possible, which was discussed in section 10.3.

10.4.2. *Separated flow analysis*

Algorithm 6 gives a generic way to compute performance bounds, taking into account the pay burst only once phenomenon. It first computes an arrival curve for each flow at each intermediate system: $\alpha_i^{(h)}$ is an arrival curve for $F_i^{(h)}$. Then, it computes the residual service curve for each flow at each system: $\beta_i^{(h)}$ is the residual service curve for flow i at system h. Finally, it computes the global service curve for each flow by a convolution, and worst-case performance upper bounds can be computed using that curve and Theorem 5.2.

Algorithm 6 is valid for blind multiplexing (and thus for any other service policy) if the systems offer strict service curves. The performance bounds computed with this algorithm can be improved if more information is known about the systems and the curves.

1) If a maximum service curve $\overline{\beta}^{(h)}$ and a shaper $\sigma^{(h)}$ is known for each server h, as well as a minimum arrival curve $\underline{\alpha}_i$ for each flow i, then lines (4–5) can be replaced by the following three lines (with obvious notations), following Theorem 5.3:

$$\begin{aligned}
&\beta_i^{(h)} \leftarrow \left[\beta^{(j)} - \textstyle\sum_{j\in \mathrm{Fl}(h)\setminus\{i\}} \alpha_j^{(\mathrm{pred}_i(h))}\right]^+ ;\\
&\alpha_i^{(h)} \leftarrow \min(\alpha_i^{(\mathrm{pred}_i(h))} * \overline{\beta}^{(\mathrm{pred}_i(h))} \oslash \beta_i^{(h)}, \sigma^{(\mathrm{pred}_i(h))});\\
&\underline{\alpha}_i^{(h)} \leftarrow \underline{\alpha}_i^{(\mathrm{pred}_i(h))} * \beta_i^{(h)}.
\end{aligned}$$

2) If the service policy is FIFO (server can offer a min-plus service curve), SP or GPS, it suffices to replace lines (4–5) with the formulas corresponding to the specific policy, i.e. using Theorems 7.5, 7.6 or 7.7. Naturally, it is possible to combine the different service policies with maximum service curves and minimum arrival curves.

Algorithm 6: Generic SFA (separated flow analysis)

```
    Data: $(\alpha_i)_{i=1}^m \in \mathcal{F}^m$, $(\beta^{(h)})_{h=1}^n \in \mathcal{F}^n$, $(\mathbf{p}_i)_{i=1}^m$ increasing paths.
    Result: $\tilde{\beta}_i$ a min-plus service curve for each flow $i$, for the global network.
1   begin
2       for $i = 1$ to $m$ do
3           $\alpha_i^{(0)} \leftarrow \alpha_i$;
4       for $h = 1$ to $n$ do
5           foreach $i$ such that $h \in \mathbf{p}_i$ do
6               $\beta_i^{(h)} \leftarrow \left[\beta^{(h)} - \sum_{j\in \mathrm{Fl}(h)\setminus\{i\}} \alpha_j^{(\mathrm{pred}_i(h))}\right]^+$;
7               $\alpha_i^{(h)} \leftarrow \alpha_i^{(\mathrm{pred}_i(h))} \oslash \beta_i^{(h)}$ ;
8       for $i = 1$ to $m$ do
9           $\tilde{\beta}_i \leftarrow *_{h\in \mathbf{p}_i} \beta_i^{(h)}$.
```

The algorithm presented here does not claim optimality, but algorithmic efficiency. Indeed, it requires $\mathcal{O}(nm)$ basic operations (convolution, deconvolution, etc.).

EXAMPLE 10.5.– *Take the example of Figure 10.1 (top) again. We obtain:*

$$\begin{cases} \alpha_1^{(0)} = \alpha_1, & \alpha_2^{(0)} = \alpha_2, \\ \alpha_3^{(0)} = \alpha_3, & \\ \beta_1^{(1)} = \left[\beta^{(1)} - \alpha_3\right]^+, & \alpha_1^{(1)} = \alpha_1 \oslash \left[\beta^{(1)} - \alpha_3\right]^+, \\ \beta_3^{(1)} = \left[\beta^{(1)} - \alpha_1\right]^+, & \alpha_3^{(1)} = \alpha_3 \oslash \left[\beta^{(1)} - \alpha_1\right]^+, \\ \beta_1^{(2)} = \left[\beta^{(2)} - \alpha_2\right]^+, & \alpha_1^{(2)} = \alpha_1^{(1)} \oslash \left[\beta^{(2)} - \alpha_2\right]^+, \\ \beta_2^{(2)} = \left[\beta^{(2)} - \alpha_1^{(1)}\right]^+, & \alpha_2^{(2)} = \alpha_2 \oslash \left[\beta^{(2)} - \alpha_1^{(1)}\right]^+, \\ \beta_2^{(3)} = \left[\beta^{(3)} - \alpha_3^{(1)}\right]^+, & \alpha_2^{(3)} = \alpha_2^{(2)} \oslash \left[\beta^{(3)} - \alpha_3^{(1)}\right]^+, \\ \beta_3^{(3)} = \left[\beta^{(3)} - \alpha_2^{(2)}\right]^+, & \alpha_3^{(3)} = \alpha_3^{(1)} \oslash \left[\beta^{(3)} - \alpha_2^{(2)}\right]^+. \end{cases}$$

If the service policy is static priority with flow 1 having the highest priority and flow 3 the lowest, then the computed arrival and service curves become:

$$\begin{cases} \alpha_1^{(0)} = \alpha_1, & \alpha_2^{(0)} = \alpha_2, \\ \alpha_3^{(0)} = \alpha_3, & \\ \beta_1^{(1)} = \beta^{(1)}, & \alpha_1^{(1)} = \alpha_1 \oslash \beta^{(1)}, \\ \beta_3^{(1)} = \left[\beta^{(1)} - \alpha_1\right]^+, & \alpha_3^{(1)} = \alpha_3 \oslash \left[\beta^{(1)} - \alpha_1\right]^+, \\ \beta_1^{(2)} = \beta^{(2)}, & \alpha_1^{(2)} = \alpha_1^{(1)} \oslash \beta^{(2)}, \\ \beta_2^{(2)} = \left[\beta^{(2)} - \alpha_1^{(1)}\right]^+, & \alpha_2^{(2)} = \alpha_2 \oslash \left[\beta^{(2)} - \alpha_1^{(1)}\right]^+, \\ \beta_2^{(3)} = \beta^{(3)}, & \alpha_2^{(3)} = \alpha_2^{(2)} \oslash \beta^{(3)}, \\ \beta_3^{(3)} = \left[\beta^{(3)} - \alpha_2^{(2)}\right]^+, & \alpha_3^{(3)} = \alpha_3^{(1)} \oslash \left[\beta^{(3)} - \alpha_2^{(2)}\right]^+. \end{cases}$$

10.4.3. *Group flow analysis*

Total output analysis and separated flow analysis consider either the whole set of flows crossing a server, or each flow individually. These are two extremal solutions, and grouping flows can be done in a more clever way. This is especially the case when a shaper $\sigma^{(h)}$ is known for each server h.

In TOA, when a flow is aggregated at a server, the whole worst-case backlogged bound is used for the next server. Moreover, it computes the worst-case delay for each server.

In SFA, a residual service curve per server and flow is obtained by using the formulas $\beta_i^{(h)} = \left[\beta^{(j)} - \sum_{j \in \mathrm{Fl}(h)\setminus\{i\}} \alpha_j^{(\ell)}\right]^+$ and $\alpha_i^{(h)} = \min(\alpha_i^{(\ell)} \oslash \beta_i^{(h)}, \sigma^{(\ell)})$, with $\ell = \mathrm{pred}_i(h)$. This method can become very pessimistic when several flows follow the same arc in the network. Indeed, consider the network of Figure 10.1 (middle) and server 2. The arrival curves $\alpha_1^{(1)} = [\beta^{(1)} - \alpha_2]^+ \wedge \sigma(1)$ and $\alpha_2^{(1)} = [\beta^{(1)} - \alpha_1]^+ \wedge \sigma(1)$ have been computed. When computing the service offered by server 2 to flow 3, we will compute $\beta_3^{(2)} = [\beta^{(2)} - \alpha_1^{(1)} - \alpha_2^{(1)}]^+$. Another solution would be to compute the global arrival curve to server 2 from server 1, which is, from Theorem 5.3, $\eta = (\alpha_1 + \alpha_2) \oslash \beta^{(1)} \wedge \sigma^{(1)}$, and the residual service curve for flow 3 is thus $[\beta^{(2)} - \eta]^+ \geqslant \beta_3^{(2)}$.

The group flow analysis principle is to compute arrival curves for flows along the arcs of the network.

The same argument can be used for maximal service curves instead of shaping curves, but to keep things simple, only shaping curves will be discussed here. Similarly, this approach could be applied to any way of grouping flows. We restrict ourselves here to the grouping along the arcs of the network. It is a natural way of grouping, which takes advantage of the shaper. This way of grouping flows can potentially lead to an algorithm with an exponential complexity.

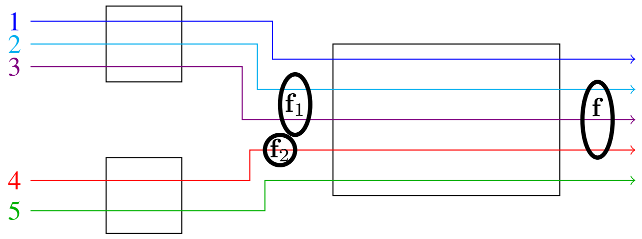


Figure 10.5. *Grouping flows according to the arcs of the network: grouping flows $\mathbf{f} = \{2, 3, 4\}$ after server h requires them to be partitioned in two subsets $\mathbf{f}_1$ and $\mathbf{f}_2$ depending on their previous visited server. For a color version of this figure, see www.iste.co.uk/bouillard/calculus.zip*

To generalize the above example, let us consider server h, depicted in Figure 10.5. To compute the residual service curve of flows in $\mathbf{f}$ and their arrival curve after server h, we need to know the arrival curve η of this group of flows and that of the other flows η' in order to use the formulas

$$\beta_{\mathbf{f}}^{(h)} = [\beta^{(h)} - \eta']^+ \qquad \text{and} \qquad \alpha_{\mathbf{f}}^{(h)} = \eta \oslash \beta_{\mathbf{f}}^{(h)} \wedge \sigma^{(h)}.$$

To compute the arrival curves η and η', it is possible to group flows according to the arc they cross. For this, let us introduce an additional notation: $\mathrm{Fl}(\ell, k)$ is the set of flows that cross servers ℓ and k consecutively (equivalently, it is the set of flows that follow the arc (ℓ, k)). Thus, we can now write the formula:

$$\eta = \sum_{\ell \mid (\ell,h)\in E} \alpha^{(\ell)}_{\mathbf{f}\cap\mathrm{Fl}(\ell,h)} + \sum_{i\in\mathbf{f} \mid \mathrm{fst}(i)=h} \alpha^{(0)}_i,$$

and similarly for η' by replacing $\mathbf{f}$ with $\bar{\mathbf{f}} = \mathrm{Fl}(h)\backslash\mathbf{f}$.

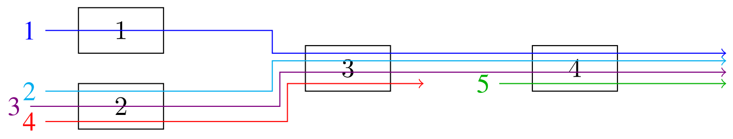


Figure 10.6. *Network example for the group flow analysis. For a color version of this figure, see www.iste.co.uk/bouillard/calculus.zip*

The group flow analysis is then performed in the following two steps, given by Algorithm 7:

1) Computing the group of flows that are needed for the analysis. This is done by a backward exploration on the servers. If the performance is computed for each flow i, we must compute the residual service curve for each flow i for each server crossed by it, hence the initialization line 2. Then, line 6 groups of flows to be evaluated at each server are computed following the example of Figure 10.5 for each server h.

2) Computing the residual service curves and arrival curves that are needed for the analysis of flow i. This is done server by server following the topological order. The first step ensures that all the needed grouping have been computed. Note that the empty set can be an element of $\mathbf{F}_h$. We use the convention that $\alpha^{(h)}_{\emptyset} = 0$.

Since the number of flow groups requested might be exponential, it might be useful to restrict to the computation of the residual service curve of a subset of flow of interest (e.g. a single flow i). In that case, only the initialization step line 2 is modified, by replacing $\mathbf{F}_h \leftarrow \cup_{i\in\mathrm{Fl}(h)}\{\{i\}\}$ with $\mathbf{F}_h \leftarrow \{\{i\}\}$ if $i \in \mathrm{Fl}(h)$ and $\emptyset$ otherwise.

EXAMPLE 10.6.– *Take the example of Figure 10.6, where the flow of interest is flow 2. At the initialization step line 2, we get* $\mathbf{F}_2 = \mathbf{F}_3 = \mathbf{F}_4 = \{\{2\}\}$ *and* $\mathbf{F}_1 = \emptyset$. *Then, backward computing of the groups gives (we do not take into account the empty sets that might appear):* $\mathbf{F}_4 = \{\{2\}\}$ *(this is never modified),* $\mathbf{F}_3 = \{\{2\}, \{1, 3\}\}$, $\mathbf{F}_2 = \{\{3\}, \{2, 4\}, \{2\}, \{3, 4\}\}$ *and* $\mathbf{F}_1 = \{\{1\}\}$.

Algorithm 7: Generic GFA (group flow analysis)

```
Data: $(\alpha_i)_{i=1}^m \in \mathcal{F}^m$, $(\beta^{(h)})_{h=1}^n \in \mathcal{F}^n$, $(\mathbf{p}_i)_{i=1}^m$ increasing paths, $E$ the set of
      arcs of $G_{\mathcal{N}}$.
Result: $\tilde{\beta}_i$ a min-plus service curve for each flow $i$ for the global network.
1  begin
2      for $h = 1$ to $n$ do $\mathbf{F}_h \leftarrow \cup_{i \in \mathrm{Fl}(h)}\{\{i\}\}$ ;
3      for $h = n$ to $1$ do
4          foreach $\mathbf{f} \in \mathbf{F}_h$ do
5              foreach $\ell$ such that $(\ell, h) \in E$ do
6                  $\mathbf{F}_\ell \leftarrow \mathbf{F}_\ell \cup \{\mathbf{f} \cap \mathrm{Fl}(\ell, h), \bar{\mathbf{f}} \cap \mathrm{Fl}(\ell, h)\}$;
7      for $h = 1$ to $n$ do
8          foreach $\mathbf{f} \in \mathbf{F}_h$ do
9              $\eta_{\mathbf{f}}^{(h)} \leftarrow \sum_{\ell \mid (\ell,h) \in E} \alpha_{\mathbf{f} \cap \mathrm{Fl}(\ell,h)}^{(\ell)} + \sum_{i \in \mathbf{f} \mid \mathrm{fst}(i)=h} \alpha_i$;
10             $\eta_{\bar{\mathbf{f}}}^{(h)} \leftarrow \sum_{\ell \mid (\ell,h) \in E} \alpha_{\bar{\mathbf{f}} \cap \mathrm{Fl}(\ell,h)}^{(\ell)} + \sum_{i \notin \mathbf{f} \mid \mathrm{fst}(i)=h} \alpha_i$;
11         foreach $\mathbf{f} \in \mathbf{F}_h$ do
12             $\beta_{\mathbf{f}}^{(h)} \leftarrow [\beta^{(h)} - \eta_{\bar{\mathbf{f}}}^{(h)}]^+$;
13             $\alpha_{\mathbf{f}}^{(h)} \leftarrow (\eta_{\mathbf{f}}^{(h)} \oslash \beta_{\mathbf{f}}^{(h)}) \wedge \sigma^{(h)}$;
14     for $i = 1$ to $m$ do
15         $\tilde{\beta}_i \leftarrow *_{h \in \mathbf{p}_i} \beta_{\{i\}}^{(h)}$.
```

Now we can compute the arrival curves and service curves:

In server 1, $\eta_{\{1\}}^{(1)} = \alpha_1$, $\beta_{\{1\}}^{(1)} = \beta^{(1)}$ *and* $\alpha_{\{1\}}^{(1)} = \alpha_1 \oslash \beta^{(1)} \wedge \sigma^{(1)}$.

In server 2, $\eta_{\{3\}}^{(2)} = \alpha_3$, $\eta_{\{2,4\}}^{(2)} = \alpha_2 + \alpha_4$, $\eta_{\{2\}}^{(2)} = \alpha_2$ *and* $\eta_{\{3,4\}}^{(2)} = \alpha_3 + \alpha_4$, *then*

$$\begin{cases} \beta_{\{3\}}^{(2)} = \beta^{(2)} - \eta_{\{2,4\}}^{(2)} & \alpha_{\{3\}}^{(2)} = \alpha_2 \oslash \beta_{\{3\}}^{(2)} \wedge \sigma^{(2)} \\ \beta_{\{2,4\}}^{(2)} = \beta^{(2)} - \eta_{\{3\}}^{(2)} & \alpha_{\{2,4\}}^{(2)} = \alpha_2 \oslash \beta_{\{2,4\}}^{(2)} \wedge \sigma^{(2)} \\ \beta_{\{2\}}^{(2)} = \beta^{(2)} - \eta_{\{3,4\}}^{(2)} & \alpha_{\{2\}}^{(2)} = \alpha_2 \oslash \beta_{\{2\}}^{(2)} \wedge \sigma^{(2)} \\ \beta_{\{3,4\}}^{(2)} = \beta^{(2)} - \eta_{\{2\}}^{(2)} & \alpha_{\{3,4\}}^{(2)} = \alpha_2 \oslash \beta_{\{3,4\}}^{(2)} \wedge \sigma^{(2)}. \end{cases}$$

In server 3, $\eta^{(3)}_{\{1,3\}} = \alpha^{(1)}_{\{1\}} + \alpha^{(2)}_{\{3\}}$, $\eta^{(3)}_{\{2,4\}} = \alpha^{(2)}_{\{2,4\}}$, $\eta^{(3)}_{\{2\}} = \alpha^{(2)}_{\{2\}}$ *and* $\eta^{(3)}_{\{1,3,4\}} = \alpha^{(1)}_{\{1\}} + \alpha^{(2)}_{\{3,4\}}$*, then*

$$\begin{cases} \beta^{(3)}_{\{1,3\}} = \beta^{(3)} - \eta^{(3)}_{\{2,4\}} & \alpha^{(3)}_{\{1,3\}} = \eta^{(3)}_{\{1,3\}} \oslash \beta^{(2)}_{\{1,3\}} \wedge \sigma^{(3)} \\ \beta^{(3)}_{\{2\}} = \beta^{(3)} - \eta^{(3)}_{\{1,3,4\}} & \alpha^{(3)}_{\{2\}} = \eta^{(3)}_{\{2\}} \oslash \beta^{(3)}_{\{2\}} \wedge \sigma^{(3)}. \end{cases}$$

Finally, in server 4, we have $\eta^{(4)}_2 = \alpha^{(3)}_{\{2\}}$ *and* $\eta_{\{1,3,5\}} = \alpha^{(3)}_{\{1,3\}} + \alpha_5$*, then*

$$\beta^{(4)}_{\{2\}} = \beta^{(4)} - \eta^{(3)}_{\{1,3,5\}} \quad \textit{and} \quad \alpha^{(4)}_{\{2\}} = \eta^{(4)}_{\{2\}} \oslash \beta^{(4)}_{\{2\}} \wedge \sigma^{(4)}.$$

10.5. NP-hardness of computing tight bounds

In the previous sections, several ways of computing worst-case performance bounds have been described. The algorithms all have a polynomial complexity in the size of the network and in the number of basic operations performed (and these operations for the usual classes of functions have a polynomial complexity). However, it has also been shown in section 10.3.1 that it is not possible to obtain tight performance bounds. Here, tight means that it is possible to find cumulative departure and arrival processes that obey all of the constraints (arrival curves, service curves), such that the maximal delay of a flow or the maximum backlog is exactly the bound to be computed with the residual service curve obtained from the algorithm.

In this section, we show that the problem of finding the exact worst-case performance bound is an NP-hard problem, meaning that unless P = NP, there is no polynomial algorithm to solve the exact worst-case performance problem in the network calculus framework.

THEOREM 10.2.– *Consider a feed-forward network where every server has an arbitrary multiplexing service policy.*

– computing the maximum backlog at a given server is NP-hard;

– computing the maximum delay of a flow is NP-hard.

PROOF.– We reduce the problem *exact three-cover* (X3C) to our problem. An instance of X3C is a collection $\mathcal{C} = \{c_1, \dots, c_{3q}\}$ of $3q$ elements and a collection $\mathcal{U} = \{u_1, \dots, u_s\}$ of s sets of three elements of $\mathcal{C}$. The problem is to decide whether there exists a cover of $\mathcal{C}$ by q elements of $\mathcal{U}$. We will reduce this problem to deciding whether a given backlog or delay can be reached in a server of a network.

From an instance of X3C, we build a network as shown in Figure 10.7. More precisely, the upper stage consists of $3q$ servers $U_1, \dots, U_{3q}$, all with strict service curve λ_1. The middle stage consists of s servers $V_1, \ldots, V_s$, all with strict service curve λ_2. Finally, the lower stage has only one server V, with service curve λ_R with $R > 3s$. There are $3s$ flows, each crossing three servers from top to bottom. Flow (i, j) crosses servers V_j, V_i and W if and only if $u_j \in v_i$. Each of those interfering flows has an arrival curve $\alpha : t \mapsto \min(t, 1)$ (note that there can be no burst).

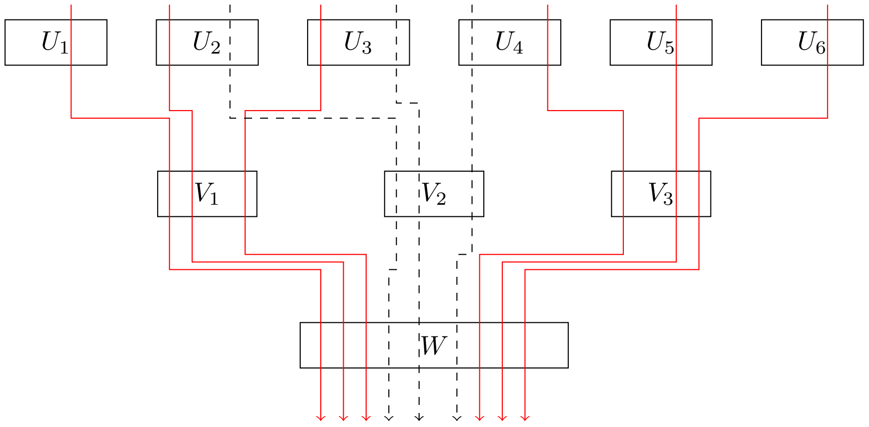


Figure 10.7. *Transformation of an instance of X3C into a network. For a color version of this figure, see www.iste.co.uk/bouillard/calculus.zip*

The example of Figure 10.7 is with $\mathcal{U} = \{1, 2, 3, 4, 5, 6\}$ and $\mathcal{V} = \{\{1, 2, 3\}, \{2, 3, 4\}, \{4, 5, 6\}\}$, so $\{\{1, 2, 3\}, \{4, 5, 6\}\}$ is a 3-cover of $\mathcal{C}$.

NP-hardness of the maximum backlog. We want to decide whether the backlog in W can be at least $3s - 2q$.

We first prove that the worst-case backlog in server W will be obtained at time $1-$ when flows arrive according to α from time 0, and when servers U_j and V_i are exact servers on $[0, 1)$ and infinite at time 1. The backlog of server W at time 1 is then the backlog in the whole system at time $1-$.

Fix 0 as the date of arrival of the first bit of data in the system.

The worst-case backlog is obtained at time $s \geqslant 1$. If it were obtained at time $s < 1$, then every bit of data would not yet have been sent into the network, and if some backlog could be created at time s, some more can be created until time 1.

The worst-case backlog is obtained at time $s = 1$. If it were obtained at time $s > 1$, consider the following transformation: delay the arrival of every flow so that every flow starts arriving at time $s - 1$. Then, the quantity of data served during $[s - 1, s)$

after transformation is less than the quantity of data served during $[0, s)$ in the initial configuration, hence the backlog has increased, and the first bit of data arrives at time $s-1$ instead of 0 and there is a worst-case trajectory with $s = 1$.

Correspondence with X3C. Consider an infinitesimal time interval, during which each flow (i, j) at the upper stage is served at rate $r_{i,j}$, with $\sum_{i:u_j \in v_i} r_{i,j} = 1$. The service rate of U_i is $\min(2, \sum_{j:u_j \in v_i} r_{i,j})$ and then the growth rate of the backlog during that time interval at the middle stage is $\sum_{i \in \{1,...s\}} (\sum_{j:u_j \in v_i} r_{i,j} - 2)_+$. At the upper stage, a backlog is created at rate $3s - \sum_{i,j} r_{ij} = 3(s-q)$, which does not depend on the flows that are served. There is no backlog created at server V since $R > 3s$. Maximizing the backlog is equivalent to maximizing the following function:

$$\sum_{i \in \{1,...s\}} [\sum_{j:u_j \in v_i} r_{i,j} - 2]^+,$$

with the constraints: $\forall j \in \{1, \ldots, 3q\}$, $\sum_{i:u_j \in v_i} r_{i,j} = 1$ and $\forall i, j$, $r_{i,j} \geqslant 0$.

Our problem boils down to the maximization of a convex function on a convex set. The maximum values are then obtained at some extremal vertices of the convex set. The extremal vertices of the convex set are such that $\sum_{i:u_j \in v_i} r_{i,j} = 1$, $r_{i,j} \in \{0, 1\}$ and then $\sum_{j:u_j \in v_i} r_{i,j} \in \{0, 1, 2, 3\}$. Maximizing our function is then equivalent to maximizing the number of i such that $\sum_{j:u_j \in v_i} r_{i,j} = 3$.

This number is upper-bounded by q, and this maximum is reached if and only if there is an X3C cover (in fact, there is a point-to-point correspondence between the set of extremal vertices that reach q and the set of X3C covers).

If there exists an X3C cover, then the backlog in the middle stage increases at rate q (at rate 1 for each server that receive data at rate 3). If there is no X3C cover, then the backlog in the middle stage increases at most at rate $q - 1$.

Finally, the above reasoning is valid for the interval of time $[0, 1)$. Indeed, the backlog is sub-additive: if an arrival cumulative function F_1 defined on the interval $[0, s]$ in a server creates a backlog b_1 at time s, and if the arrival cumulative function F_2 such that $F_2(s) = 0$ creates a backlog b_2 at time $t \geqslant s$ (when the server is empty at time s), then the process $F_1 + F_2$ creates a backlog $b \leqslant b_1 + b_2$ at time t. As a result, there is no advantage to changing the service rates $r_{i,j}$ during the interval of time $[0, 1)$. At time 1, servers U_j and V_i serve all of their backlog and the backlog in server W is then at most $3s - 2q$. This maximum is reached if and only if there is an X3C cover. If there is no X3C cover, then the backlog is at most $3s - 2q - 1$.

NP-hardness of the delay. We keep the same scheme of reduction. There is an additional flow crossing server W only and we are now interested in computing the worst-case delay for a bit of data of that flow (we can take λ_r with $r < R - 3s$ and the maximum delay can be obtained for the first bit of data arriving in the system).

The worst-case scenario for that network first creates a backlog in server W and then considers every other server as an infinite capacity server. To create the backlog, we use the previous construction.

Backlog at a server of the upper stage can be created if data arrive at a rate more than 1.

Backlog at a server of the middle stage can be created if there is an arrival rate of $r > 2$ at a server of the middle stage. The backlog grows at rate $r - 2$. If the arrival rate is exactly 3, then the flows arrive at rate 1, blocking the flows at the upper level, and so this is the scenario that builds the greatest backlog.

Suppose that exactly k servers of the middle stage have an arrival rate of 3 between time 0 and t_k. Without loss of generality, the servers that create backlog are $V_1, \dots, V_k$. The backlog created is $(N_k - 2k)t_k$, where $N_k = |\{(\ell, j) \mid \exists i \leqslant k,\ u_j \in v_i \cap v_\ell\}|$ and for the other flows, either they are idle or data are transmitted. For these other flows, no backlog is created at the middle stage, but some backlog can be built at the upper stage at rate $[\sum_{\ell\,:\,u_j \in v_\ell} r_{\ell,j} - 1]^+$ for each server C_j. We can upper-bound the backlog created by using the inequality $N_k \leqslant 3s - 3(q-k)$ and lower-bounding the remaining flows by $3(q-k)$, and consider them as idle (they will start transmitting data at time t_k). We note that if $k = q$, this is not an approximation.

As a result, at time $t \geqslant t_k$, the quantity of data that arrived at server W between t_k(start of the backlogged period) and t is at most

$$Q_k(t) = (3s - 3q + k)t_k + 3s(\min(t,1) - t_k) + 3(q-k)(\max(t,1) - 1).$$

The delay is thus less than $t - t_k$ such that $Q_k(t) = R(t - t_k)$.

If $t \leqslant 1$, then $t - t_k = \frac{3(s-q)+k}{R-3s} t_k$, and $t - t_k$ is increasing with t_k. If $t \geqslant 1$, then $t - t_k = \frac{3(s-q-k)-2kt_k}{R-3(q-k)}$, and $t - t_k$ is decreasing with t_k. Therefore, to obtain a maximum delay, t_k has to be chosen so that $t = 1$. Then, $t_k = \frac{R-3s}{R-3q+k}$ and $d_k = \frac{3(s-q)+k}{R-3q+k}$ is an upper bound of the delay. As $R > 3s$, d_k increases with k. d_q can effectively be achieved for $k = q$ as there is no approximation in that specific case.

If there exists an X3C cover, then the maximum delay is $\frac{3s-2q}{R-2q}$, and if there is no X3C cover, then the maximum delay that can be achieved is $\frac{3s-2q-1}{R-2q-1}$. Therefore, the question of whether a delay greater or equal to $\frac{3s-2q}{R-2q}$ can be achieved is NP-hard. □

10.6. Conclusion

Although several results exist to compute *exact* worst-case performances when considering a single server, as mentioned in Theorem 5.2 and in the paragraph on tightness after Theorems 7.1 and 7.4 (sections 7.2.1 and 7.3.2.1), the problem is more complex when considering a whole network.

This chapter started with an illustration of the main sources of loss of tightness while considering a sequence of servers. Then, three algorithms handling general feed-forward topologies were presented. Finally, the general result showed that computing exact worst-case bound is NP-hard in the general case.

We note that the definition of algorithms based on iterative application of local results is an active area of research [BON 16c, BON 17b], and the three algorithms presented are more instructive than up to date. Computing accurate bounds while using residual service curves requires having different points of view on the network: cutting single flow into a sequence of two flows [LEN 08], extending a flow [BON 17a], maximizing flow aggregation of cross-traffic [BON 16a] and so on.

For an evaluation of the pessimism of network calculus, we may have to look into different studies, both industrial [BOY 12b] and academic [BON 16c, BON 17b].

11
Tight Worst-case Performances

In the previous chapter, we described several algorithms to compute performances in feed-forward networks. All these algorithms share some common characteristics: first, they are based on an exploration of the servers from the sources of the flows to their destination. Second, residual service curves and output arrival curves are computed for each flow (or group of flows) and each server.

In section 10.3.1, examples are given showing that the bounds computed with these methods cannot be tight, because they are based on computing residual curves.

The aim of this chapter is to present methods to compute tight performance bounds, by using a different paradigm from that in the previous chapter: we describe the network by a linear program that (1) mimics the exploration of the network from the destination (of the flow of interest) to the sources and (2) describes the characteristics of an admissible trajectory (i.e. the possible cumulative functions given arrival and service curves). As a result, no residual service curve or additional arrival curve is computed.

This technique can be interpreted as an alternative and more precise generalization of the pay multiplexing only once phenomenon than Theorem 10.1: in section 10.3.1 an example is provided where, depending on the parameters, PMOO leads to better or worse performances. Here, in a sense, the linear problem can make the decision on what will lead to the best computation.

In this chapter, we focus exclusively on tandem networks, but it could also be relevant to the more general context of feed-forward networks (at an exponential computational cost). First, in section 11.1, we study the case of arbitrary multiplexing and show that the exact worst-case delay bound can be computed by a linear program with a polynomial number of variables and constraints. In section 11.2, we focus on

Deterministic Network Calculus: From Theory to Practical Implementation,
First Edition. Anne Bouillard, Marc Boyer and Euriell Le Corronc.
© ISTE Ltd 2018. Published by ISTE Ltd and John Wiley & Sons, Inc.

the FIFO policy. There, the number of variables becomes exponential and Boolean variables need to be introduced.

11.1. Tandem networks under arbitrary multiplexing

11.1.1. *Example of two servers in tandem*

Consider the network shown in Figure 10.2 with $(A_1, A_2) \xrightarrow{\mathcal{S}_1} (B_1, B_2) \xrightarrow{\mathcal{S}_2} (C_1, C_2)$, leaky buckets arrival curves ($\alpha_i = \gamma_{r_i,b_i}$) and rate-latency strict service curves ($\beta^{(h)} = \beta_{R_h,T_h}$). Before making any approximation, let us write the equations that are used to compute a worst-case delay upper bound. We assume that t is the time at which a bit of data of flow 2 suffering the worst-case delay exits the network for this flow. We call this bit of data the *bit of interest*. We have

$$\begin{cases} C_1(t) + C_2(t) \geqslant C_1(s) + C_2(s) + \beta^{(2)}(t-s) & \forall s \in [\mathit{Start}_2(t), t], \\ B_1(u) + B_2(u) \geqslant B_1(v) + B_2(v) + \beta^{(1)}(u-v) & \forall v \in [\mathit{Start}_1(u), u], \\ A_1(w) - A_1(v) \leqslant \alpha_1(w-v) & \forall v, w \in [0, u], \\ A_2(w) - A_2(v) \leqslant \alpha_2(w-v) & \forall v, w \in [0, u]. \end{cases}$$

Remember that the functions are non-decreasing and that $A_i \geqslant B_i \geqslant C_i$, $i \in \{1, 2\}$.

As in the previous chapters, we can specify those equations by choosing s, u and v. A natural choice is $s = \mathit{Start}_2(t)$, $u = s$ and $v = \mathit{Start}_1(u)$, so that $C_i(s) = B_i(s)$ and $B_i(v) = A_i(v)$ and the first two equations share a common date.

The equations then become, when we replace the service and arrival curves by their expression,

$$\begin{cases} C_1(t) + C_2(t) \geqslant B_1(s) + B_2(s) + R_2\,[t-s-T_2]^+ & \text{with } s = \mathit{Start}_2(t), \\ B_1(s) + B_2(s) \geqslant A_1(v) + A_2(v) + R_1\,[s-v-T_1]^+ & \text{with } v = \mathit{Start}_1(s), \\ A_i(s) - A_i(v) \leqslant b_i + r_i(s-v) & \text{with } i \in \{1,2\}, \\ A_i(t) - A_i(v) \leqslant b_i + r_i(t-v) & \text{with } i \in \{1,2\}, \\ A_i(t) - A_i(s) \leqslant b_i + r_i(t-s) & \text{with } i \in \{1,2\}. \end{cases}$$

Note that the first two lines can be replaced by

$$\begin{cases} C_1(t) + C_2(t) \geqslant B_1(s) + B_2(s) + R_2(t-s-T_2) & \text{with } s = \mathit{Start}_2(t), \\ C_1(t) + C_2(t) \geqslant B_1(s) + B_2(s), \\ B_1(s) + B_2(s) \geqslant A_1(v) + A_2(v) + R_1(s-v-T_1) & \text{with } v = \mathit{Start}_1(s), \\ B_1(s) + B_2(s) \geqslant A_1(v) + A_2(v). \end{cases}$$

We can also translate the monotony and the causality at dates v, s, t into the following inequalities (we only need to define B_i at s and C_i at t):

$$\begin{cases} A_i(s) \geqslant B_i(s) \text{ and } A_i(t) \geqslant C_i(t) & \text{for } i \in \{1, 2\}, \\ A_i(t) \geqslant A_i(s) \geqslant A_i(v) & \text{for } i \in \{1, 2\}. \end{cases}$$

Let us denote by z the arrival date of the bit of data of interest. We must have $A_2(z) \leqslant C_2(t) \leqslant A_2(z+)$ (the bit of interest could arrive in a burst or, more specifically here, be the last bit of data of a burst).

In order to compute a bound for the worst-case delay of flow 2, it then suffices to solve those equations. It should be noted that if we accept replacing $A_2(z) \leqslant C_2(t) \leqslant A_2(z+)$ by $A_2(z) = C_2(t)$, all of the inequalities are linear and thus can be solved by a linear program. This linear program is given in Table 11.1. Here and in the rest of the chapter, we differentiate the variables from the quantity they represent (which can be a date or the value of a function at a date) by writing the variables in bold font and forgetting the parenthesis. For example, variable $\mathbf{t}$ is the variable representing date t and variable $\mathbf{A_1t}$ is the variable representing $A_1(t)$.

Objective	Maximize $\mathbf{t} - \mathbf{v}$
Under the constraints	
Dates	$\mathbf{t} \geqslant \mathbf{s} \geqslant \mathbf{v} \quad \mathbf{t} \geqslant \mathbf{z} \geqslant \mathbf{v}$
Monotonicity	$\mathbf{A_it} \geqslant \mathbf{A_is} \geqslant \mathbf{A_iv},\ i \in \{1,2\} \quad \mathbf{A_1s} \geqslant \mathbf{A_1z}$
Causality	$\mathbf{A_is} \geqslant \mathbf{B_is},\ \mathbf{A_it} \geqslant \mathbf{C_it}$
Arrival	$\mathbf{A_iy} - \mathbf{A_ix} \leqslant b_i + r_i\mathbf{y} - r_i\mathbf{x},$
	$(\mathbf{x}, \mathbf{y}) \in \{(\mathbf{v}, \mathbf{s}), (\mathbf{v}, \mathbf{z}), (\mathbf{v}, \mathbf{t}), (\mathbf{z}, \mathbf{t}), (\mathbf{s}, \mathbf{t})\}$
Service	$\mathbf{C_1t} + \mathbf{C_2t} \geqslant \mathbf{B_1s} + \mathbf{B_2s} + R_2\mathbf{t} - R_2\mathbf{s} - R_2T_2$
	$\mathbf{C_1t} + \mathbf{C_2t} \geqslant \mathbf{B_1s} + \mathbf{B_2s}$
	$\mathbf{B_1s} + \mathbf{B_2s} \geqslant \mathbf{A_1v} + \mathbf{A_2v} + R_1\mathbf{s} - R_1\mathbf{v} - R_1T_1$
	$\mathbf{B_1s} + \mathbf{B_2s} \geqslant \mathbf{A_1v} + \mathbf{A_2v}$

Table 11.1. *Linear program for computing the worst-case delay in a network of two servers in tandem*

In fact, this linear program gives the exact worst-case delay of the network, as we will prove in the following section.

11.1.2. *A linear program for tandem networks*

We first present the general scheme of transformation of a tandem network into a linear program before proving that its optimal solution is the exact worst-case delay of the network. Note that these results can be generalized at no cost to tree networks and adapted to the computation of the maximum backlog at a server.

Worst-case performance bounds can be computed using the linear program under the following assumptions on the arrival and service curves:

– arrival curves are concave and piecewise linear (we can write $\alpha_i = \bigwedge_k \gamma_{r_k,b_k}$);

– strict service curves are convex and piecewise linear ($\beta^{(h)} = \bigvee_k \beta_{R_k,T_k}$).

We assume without loss of generality that flow 1 is the flow of interest and that this flow departs the network at server n.

11.1.2.1. *Variables*

There are two types of variables, namely time variables, which represent dates, and function variables, which represent the values of the functions at some of these dates:

– *time variables*: $\mathbf{t_h}$ for all $h \in \{0, \dots, n\}$, as well as a variable $\mathbf{u}$. Interpretation: $\mathbf{t_n}$ represents the instant at which the worst case occurs: it is the departure time of the bit of data that suffers the maximal delay. Then, $\forall h < n$, $\mathbf{t_h} = \mathit{Start}_{h+1}(\mathbf{t_{h+1}})$, the start of the backlogged period of $\mathbf{t_{h+1}}$ at the server h. Variable $\mathbf{u}$ represents the arrival time u of the bit of data suffering the maximal delay in the first server;

– *function variables (departure processes)*: $\mathbf{F_i^{(h)} t_h}$ and $\mathbf{F_i^{(h)} t_{h-1}}$ for all $i \in \{1, \dots, m\}$ and $h \in \mathbf{p}_i$. Interpretation: represents the value of the departure cumulative function of flow i at server h (or equivalently if flow i crosses server $h+1$, the arrival cumulative function of flow i at server $h+1$) at times t_h and t_{h-1};

– *function variables (arrival processes)*: $\mathbf{F_i^{(\mathrm{fst}(i)-1)} t_h}$ for all $i \in \mathcal{F}$, and $h \in \mathbf{p}_i$. Interpretation: represents the value of the cumulative function $F_i^{(\mathrm{fst}(i)-1)}$ at time t_h.

11.1.2.2. *Linear constraints*

The linear constraints can be intuitively deduced from the interpretation of the variables:

– *time constraints*: $\{\mathbf{t_h} \leqslant \mathbf{t_{h+1}} \mid h < n\}$;

– *starts of backlogged periods*: $\{\mathbf{F_i^{(h-1)} t_{h-1}} = \mathbf{F_i^{(h)} t_{h-1}} \mid 1 \leqslant i \leqslant m,\ h \in \mathbf{p}_i\}$;

– *strict service constraints*: $\{\sum_{i \in \mathbf{p}_h} (\mathbf{F_i^{(h)} t_h} - \mathbf{F_i^{(h)} t_{h-1}}) \geqslant \beta^{(h)}(\mathbf{t_h} - \mathbf{t_{h-1}}) \mid 1 \leqslant h \leqslant n\}$ (this inequality represents several linear inequalities, since $\beta^{(h)}$ is the maximum of linear functions);

– *causality constraints*: $\{\mathbf{F_i^{(\mathrm{fst}(i)-1)} t_h} \geqslant \mathbf{F_i^{(h)} t_j} \mid 1 \leqslant i \leqslant m,\ h \in \mathbf{p}_i\}$;

– *non-decreasing functions*: $\{\mathbf{F_i^{(h)}t_{h-1}} \geqslant \mathbf{F_i^{(h)}t_h} \mid 1 \leqslant i \leqslant m,\ h \in \mathbf{p}_i\}$ and $\{\mathbf{F_i^{(fst(i)-1)}t_{h-1}} \geqslant \mathbf{F_i^{(fst(i)-1)}t_h} \mid 1 \leqslant i \leqslant m,\ h \in \mathbf{p}_i\}$;

– *arrival constraints*: $\{\mathbf{F_i^{(fst(i)-1)}t_h} - \mathbf{F_i^{(fst(i)-1)}t_\ell} \leqslant \alpha_i(\mathbf{t_h} - \mathbf{t}_\ell) \mid 1 \leqslant i \leqslant m,\ \mathrm{fst}(i) - 1 \leqslant h < \ell \leqslant \mathrm{end}(i)\}$ (this inequality represents several linear inequalities, since α_i is the minimum of linear functions);

– *insertion of* u: $\{\mathbf{t}_{\mathrm{fst}(\mathbf{i})-\mathbf{1}} \leqslant \mathbf{u} \leqslant \mathbf{t_n},\ \mathbf{F_1^{(fst(1)-1)}u} > \mathbf{F_1^{(n)}t_n}, \mathbf{F_1^{(fst(1)-1)}u} - \mathbf{F_1^{(fst(1)-1)}}\mathbf{t}_{\mathrm{fst}(\mathbf{1})-\mathbf{1}} \leqslant \alpha_1(\mathbf{u} - \mathbf{t}_{\mathrm{fst}(\mathbf{1})-\mathbf{1}})\}$.

11.1.2.3. *Objective*

Depending on the worst-case performance to compute, two different objectives can be used:

– *worst end-to-end delay for flow* 1: maximize $\mathbf{t_n} - \mathbf{u}$;

– *worst backlog at server* n: maximize $\sum_{i\in\mathrm{Fl}(n)} \mathbf{F_i^{(fst(i)-1)}t_n} - \mathbf{F_i^{(n)}t_n}$. Note that intuitively one should have $\mathbf{F_i^{(n-1)}t_n}$ instead of $\mathbf{F_i^{(fst(i)-1)}t_n}$, but this variable is not defined in our linear program. In fact, the worst-case backlog at server n is obtained when all the other servers serve instantaneously their backlog, which corresponds to the objective. When computing the worst-case backlog, variable $\mathbf{u}$ and the related linear constraints can be dropped.

11.1.3. *The equivalence theorem*

THEOREM 11.1.– *Let $\mathcal{N}$ be a tandem network with n servers and m flows. Then, given a flow i (respectively a server j), the linear program described above has $\mathcal{O}(nm)$ variables and $\mathcal{O}(mn^2)$ constraints and the optimum is the worst end-to-end delay for flow i (respectively the worst backlog at server j).*

PROOF.– There are $n + 2$ time variables; the number of variable indices by flow i is linear in the length of its path, then in $\mathcal{O}(n)$. Hence, the number of variables is in $\mathcal{O}(mn)$. The number of constraints of each type is in $\mathcal{O}(mn^2)$ (this order is achieved for the arrival curve constraints).

Let D be the optimal solution of the linear program for a tandem network.

There are two steps in the rest of the proof: first, we prove that for any admissible trajectory with maximum delay d, we can set variables satisfying all the constraints (hence $d \leqslant D$), and the problem has value d; second, we prove that from a solution to the linear problem, we can reconstruct an admissible trajectory (hence $d \geqslant D$).

First, consider an admissible trajectory and then set t_n the departure time of the bit of data and u its arrival time. Set $\mathbf{t_n} = t_n$; from the value assigned to $\mathbf{t_n}$ and the interpretation of the variables, we can deduce the assignment of any other variable except those involving u. Set $\mathbf{u} = u$ and $\mathbf{F_1^{(fst(1)-1)}u} = F_1^{(\mathrm{fst}(1)-1)}(u+)$. As the trajectory is admissible and the constraints only encode network calculus constraints, it should be clear to the reader that every linear constraint is satisfied, and the delay of the bit of data is $t_n - u$.

Second, consider a solution to the linear program:

– the arrival cumulative process of flow i, $F_i^{(\mathrm{fst}(i)-1)}$, is defined by taking the largest non-decreasing function that is α_i-constrained and takes values $\mathbf{F_i^{(fst(i)-1)}t_h}$ at time $\mathbf{t_h}$ (note the special case for flow 1), and we only consider dates $t_{\mathrm{fst}(1)-1}$ and u;

– the departure cumulative process $F_i^{(h)}$ is defined by having $F_i^{(h)}(t) = F_i^{(\mathrm{fst}(i)-1)}(t)$ for all $t > \mathbf{t_h}$, $F_i^{(h)}(t) = F_i^{(h-1)}(t)$, for all $t < \mathbf{t_{h-1}}$ and the linear interpolation $F_i^{(h)}(t) = \mathbf{F_i^{(h)}t_{h-1}} + (t - \mathbf{t_{h-1}})\frac{\mathbf{F_i^{(h)}t_h - F_i^{(h)}t_{h-1}}}{\mathbf{t_h - t_{h-1}}}$ otherwise.

We now prove that the trajectory defined by these processes is admissible. By construction, these functions are non-decreasing, we have $F_i^{(h)} \leqslant F_i^{(h-1)}$ and the arrival processes are constrained by their arrival curves. Then, it remains to show that $\beta^{(h)}$ is a strict service curve for server h. Let F be the aggregated departure process from server h. The strict service property has to be checked only on $(\mathbf{t_{h-1}}, \mathbf{t_h}]$ as it is the only backlogged period. In this interval, the processes are linear, so there exists H such that $F(t) - F(s) = H \cdot (t - s)$. But as $\beta^{(h)}$ is convex and $\beta^{(h)}(\mathbf{t_h} - \mathbf{t_{h-1}}) = H \cdot (\mathbf{t_h} - \mathbf{t_{h-1}})$, we have $H(t - s) \geqslant \beta^{(h)}(t - s)$, hence the result. □

11.2. Tandem networks under the FIFO policy

In this section, we specialize the results with FIFO servers. Unfortunately, the linear program that we will obtain will not be solvable in polynomial time: the number of variables becomes exponential in the size of the network, and some of these variables are Boolean. However, this linear program will allow us to derive *upper* and *lower* bounds for the worst-case delay.

11.2.1. *Single node*

Let us first focus on the simple yet meaningful example of one server offering a min-plus service curve β crossed by two flows, whose respective arrival and departure processes are denoted by A_1, A_2 and D_1, D_2. For each departure date t_1 of a bit, there exists another date t_2 when that bit arrived. From Definition 7.7, if functions A_i are continuous at t_2, then $A_1(t_2) = D_1(t_1) \Leftrightarrow A_2(t_2) = D_2(t_1)$. Moreover, from

Definition 5.3, $(D_1 + D_2)(t_1) \geqslant \beta * (A_1 + A_2)(t_1)$. If β is convex, it is continuous, and then, there exists t_3 such that $D_1(t_1) + D_2(t_1) \geqslant (A_1(t_3) + A_2(t_3)) + \beta(t_1 - t_3)$. As $\beta \geqslant 0$, we can set $t_3 \leqslant t_2$. Then, the worst-case delay, with a rate-latency service curves and token-buckets arrival curves, is computed by solving the problem of Table 11.2. We use the same convention as in the previous section, and bold letters represent variables of the linear program.

Objective	Maximize $\mathbf{t_1} - \mathbf{t_2}$
Under the constraints	
Dates	$\mathbf{t_1} \geqslant \mathbf{t_2} \geqslant \mathbf{t_3}$
Monotonicity	$\mathbf{A_i t_1} \geqslant \mathbf{A_i t_2},\ i \in \{1, 2\}$
Arrival	$\mathbf{A_i t_2} - \mathbf{A_i t_3} \leqslant b_i + r_i \mathbf{t_2} - r_i \mathbf{t_3},\ i \in \{1, 2\}$
Service	$\mathbf{D_1 t_1} + \mathbf{D_2 t_1} \geqslant \mathbf{A_1 t_3} + \mathbf{A_2 t_3} + R\mathbf{t_1} - R\mathbf{t_3} - RT$
FIFO	$\mathbf{D_i t_1} = \mathbf{A_i t_2},\ i \in \{1, 2\}$

Table 11.2. *Linear program for computing the worst-case delay in an FIFO server*

In Table 11.2, we use very simple curves, but these can be generalized to piecewise linear concave arrival curves and piecewise linear convex service curves by writing several arrival and service constraints.

11.2.2. *Tandem networks*

We now consider a tandem of n servers, with flow 1 as the flow of interest, traversing every server. Let us first focus on server n. Most of the constraints of Table 11.2 can be written down, generalized to the set of all flows crossing server n. However, arrival constraints cannot be formulated for flows starting before server n. Therefore, given a departure date at server n, t_1, we can compute two input dates, related to the FIFO and service curve constraints at server n, t_2 and t_3. Those dates also describe the *output* of server $n - 1$. Thus, we can iterate the previous reasoning at server $n - 1$, for each of the two dates: the bit that exits server $n - 1$ at time t_2 entered that server at time t_4, and t_5 is a time instant in the past that verifies the service curve constraint formulated at time t_2. Similarly, for date t_3 at server $n - 1$, we can identify dates t_6 and t_7, respectively. It is easy to see that we can go backwards until server 1, doubling the number of dates and constraints at each node and adding arrival constraints whenever we hit the ingress node of a flow. In this way, we eventually set up a set of constraints that relate the dates of departure (at server n) and arrival of a bit of the tagged flow. Thus, we compute its worst-case delay by solving the linear problem that maximizes the above difference under these constraints.

More formally, the variables of our problem are the following:

– *time variables*: $\mathbf{t_1}, \ldots, \mathbf{t_{2^{n+1}-1}}$, where $\mathbf{t_{2k}}$ and $\mathbf{t_{2k+1}}$ correspond to the FIFO and the service curve constraints with regard to $\mathbf{t_k}$;

– *function values*: $\mathbf{F_i^{(h)} t_k}$ for $\mathrm{fst}(i) - 1 \leqslant h \leqslant \mathrm{end}(i)$ and $k \in [2^{n+1-h}, 2^{n+2-h} - 1]$.

The number of dates (hence of variables) grows exponentially with the tandem length, since it doubles at each node as we go backwards. Unfortunately, in a multi-server scenario, these dates are only partially ordered. Following the argument for the single server case and the network calculus constraints, we have:

– (*) for $k < 2^n$, $t_{2k+1} \leqslant t_{2k} \leqslant t_k$;

– (**) if $2^h \leqslant k, k' < 2^{h+1}$ and if $t_k \leqslant t_{k'}$, then $t_{2k} \leqslant t_{2k'}$ (cumulative functions are non-decreasing) and $t_{2k+1} \leqslant t_{2k'+1}$ (same as above, plus $\beta^{(h)}$ is convex).

These relations and the transitivity only lead to a partial order of $t_{2^h}, \ldots, t_{2^{h+1}-1}$. For example, for a two-node tandem, we have $t_1 \geqslant t_2 \geqslant t_3$, and $t_4 \geqslant t_5$, $t_4 \geqslant t_6$, $t_5 \geqslant t_7$ and $t_6 \geqslant t_7$. However, t_5 and t_6 cannot be ordered. A partial order creates a problem. Given a partial order of dates, we can enforce the same partial order on the corresponding function values, but this is not sufficient to ensure the monotonicity of the cumulative arrival and departure processes. In fact, it may well happen that, in the above example, the optimal solution to the linear program is for $t_5 < t_6$ and $F_{(i)}^{(h)}(t_5) > F_{(i)}^{(h)}(t_6)$, for some i, h, which violates monotonicity. In this case, the solution does not correspond to any feasible scenario and hence it is not the worst-case delay. To ensure monotonicity in a linear programming approach, we need to introduce Boolean variables.

LEMMA 11.1.– *Consider the following constraints:*

$$\begin{aligned}
&x_1 + (1-b)M \geqslant x_2\\
&x_2 + bM \geqslant x_1\\
&y_1 + (1-b)M \geqslant y_2\\
&y_2 + bM \geqslant y_1\\
&0 \leqslant x_1, x_2, y_1, y_2 \leqslant M\\
&b \in \{0, 1\}
\end{aligned}$$

Then, $x_1 < x_2 \Rightarrow y_1 \leqslant y_2$ *and* $x_2 < x_1 \Rightarrow y_2 \leqslant y_1$.

PROOF.– The inequality $x_1 < x_2$ imposes that $b = 0$ and then that $y_2 \geqslant y_1$. Similarly, $x_2 < x_2$ imposes that $b = 1$ and that $y_1 \geqslant y_2$. □

This lemma can be used on all dates $t_k, t_{k'}$ on which cumulative processes are defined. Since a date corresponds to a unique node in the tandem, for a date t_k, there exists a unique h such that $F_i^{(h)}(t_k)$ is defined; hence, it is sufficient to generate total orders on $t_{2^h}, \dots, t_{2^{h+1}-1}$ for each $h \in [1, n+1]$.

Using the results of Chapter 10, the constant M is easy to find: it suffices to find an upper bound of the delay and of the cumulative arrival functions. We will not comment further on this value, but assume for every variable $\mathbf{x}$ the constraint $0 \leqslant \mathbf{x} \leqslant M$.

We are now ready to define the constraints of a mixed-integer linear program (MILP) in order to compute the worst-case delay of flow 1. In the following, the expression $x \leqslant_b y$ stands for $x + (1-b)M \geqslant y; y + bM \geqslant x$.

For all $h \in \{1, \dots, n+1\}$, for all $k, k' \in \{2^{n+1-h}, \dots 2^{n+1-h} - 1\}$ and for all $i \in \{1, \dots, m\}$ such that $h \in \mathbf{p}_i$:

– *time constraints*: $\{\mathbf{t_k} \leqslant_{\mathbf{b_{k,k'}}} \mathbf{t_{k'}}\}$;

– *partial order constraints*: $\{\mathbf{b_{2k,2k+1}} = 0\}$ and $\{\mathbf{b_{2k,2k'}} = \mathbf{b_{2k+1,2k'+1}} = \mathbf{b_{k,k'}}\}$;

– *monotonicity*: $\{\mathbf{F_i^{(h)}t_k} \leqslant_{\mathbf{b_{k,k'}}} \mathbf{F_i^{(h)}t_{k'}}\}$;

– *FIFO hypothesis*: $h > \mathrm{fst}(i) \Rightarrow \{\mathbf{F_i^{(h)}t_k} = \mathbf{F_i^{(h-1)}}(\mathbf{t_{2k}})\}$;

– *service constraint*: $\{\sum_{i \in \mathrm{Fl}(h)} \mathbf{F_i^{(h)}t_k} \geqslant \sum_{i \in \mathrm{Fl}(h)} \mathbf{F_i^{(h-1)}t_{2k+1}} + \beta^{(h)}(\mathbf{t_k} - \mathbf{t_{2k+1}})\}$;

– *arrival constraints*: if $h = \mathrm{fst}(i)$, $\left\{\mathbf{F_i^{(h)}t'_k} - \mathbf{F_i^{(h)}t_k} \leqslant_{\mathbf{b_{k,k'}}} \alpha_i(\mathbf{t_{k'}} - \mathbf{t_k})\right\}$.

The objective of the linear program is then $\max \mathbf{t_1} - \mathbf{t_{2^N}}$.

THEOREM 11.2.– *The worst-case delay for flow* 1 *is the optimal solution of the mixed-integer linear program just described.*

PROOF.– Set d the worst-case delay for an admissible trajectory of the network and D the optimal solution of the MILP. The proof is in two steps: we first prove that every admissible trajectory is a solution of the MIPL, so that $d \leqslant D$. Then, we show that from any solution of the MILP, we can construct an admissible trajectory, so that $D \leqslant d$.

First, consider an admissible trajectory and set t_1 the departure time of the bit of data of interest. Set $\mathbf{t_1} = t_1$. By definition, cumulative functions are left-continuous. Then, by Proposition 3.10, we can find a date t_3 such that $\sum_{i \in \mathrm{Fl}(n)} F_i^{(n)}(t_1) \geqslant \sum_{i \in \mathrm{Fl}(n)} F_i^{(n-1)}(t_3) + \beta^{(n)}(t_1 - t_3)$ and a date t_2 such that for all flow i crossing server n, $F_i^{(n-1)}(t_2) \leqslant F_i^{(n)}(t_1) \leqslant F_{(i,N)}^{(n-1)}(t_2+)$. We can then set the variables as $\mathbf{t_k} = t_k$ and $\mathbf{F_i^{(n-1)}t_k} = F_i^{(n-1)}(t_k)$ but for $\mathbf{F_i^{(n-1)}t_2} = F_n^{(n)}(t_1)$.

By induction, we can go on assigning dates and variables $\mathbf{F_i^{(h)}t_k}$: if variables $\mathbf{t_k}$ and $\mathbf{F_{(i)}^{(h)}t_k}$ are assigned, we can assign $\mathbf{t_{2k}}$, $\mathbf{t_{2k+1}}$, $\mathbf{F_i^{(h-1)}t_{2k}}$ and $\mathbf{F_i^{(h)}t_{2k+1}}$ if $h-1 \in \mathbf{p}_i$ the same way as with $k=1$. Remark that $\mathbf{F_i^{(h-1)}t_{2k}} = \mathbf{F_i^{(h)}t_k}$.

The reason why we cannot define t_{2k} directly by $F_i^{(h-1)}(t_{2k}) = F_i^{(h)}(t_k)$ is that the functions are not necessarily continuous. As a result, it may be the case that several dates defined have the same value, with different values for the functions. This is not a problem for mathematical programs. Also note that this assignment of variables also defines a total order of the dates, which allows us to assign variables $\mathbf{b_{k,k'}}$, so that the monotonic and arrival constraints are satisfied. As a result, the assignment of the variables respects all the constraints and they are a solution to the problem and $d \leqslant D$.

On the other hand, consider a solution of the MILP. We will define a scenario that verifies all the constraints. Assign dates $t_k = \mathbf{t_k}$ for all k, fix i and h and consider the variables of the form $\mathbf{F_i^{(h)}t_k}$ and $\mathbf{t_k}$ for which $\mathbf{F_i^{(h)}t_k}$ is defined. Let $T_k = \{t_\ell \mid t_\ell = t_k\}$. We set $F_i^{(h)}(t_k) = \min_{t\in T_k} \mathbf{F_i^{(h)}t}$ and $F_i^{(h)}(t_k+) = \max_{t\in T_k} \mathbf{F_i^{(h)}t}$. If $h = \mathrm{fst}(i)-1$, then from those values, we extrapolate $F_i^{(h)}$ as the largest non-decreasing function that is α_i-constrained and takes values $\mathbf{F_i^{(\mathrm{fst}(i)-1)}(t_k)}$ at time t_k. If $h \geqslant \mathrm{fst}(i)$, then we extrapolate

$$F_i^{(h)}(t) = \min[F_i^{(h-1)}(t), \min_{t_k \geqslant t} F_i^{(h)}(t_k)].$$

We can check that:

– those functions are wide-sense increasing (because of the Boolean variables);

– arrival functions are α-constrained (by construction);

– the FIFO order is preserved: it is preserved for the dates defined by the linear program, and outside of those dates, the traffic is only made up of bursts;

– the service constraints are met by construction.

We have then built a scenario that verifies all the constraints and that has the same delay as the solution of the linear program. Then, $d \geqslant D$. □

We remark that Theorem 11.2 can be generalized to compute the worst-case delay of any flow i by only modifying the objective function to maximize $\mathbf{t_{2^{n-\mathrm{end}(i)}}} - \mathbf{t_{2^{n-\mathrm{fst}(i)+1}}}$.

11.2.3. *Bounds on the worst-case delay*

As already pointed out, computing the worst-case delay in a tandem network requires an exponential number of (potentially Boolean) variables, which makes this solution intractable for large networks.

One solution to simplify the problem (which will remain exponential) is to relax some constraints, especially the Boolean ones. An *upper bound* can be found by giving up total ordering. We only keep the partial date and function value ordering and solve one linear program, forgetting all the Boolean variables.

Similarly, any solution of the MILP can be extrapolated into an admissible trajectory and hence is a *lower bound* on the worst-case delay. An algorithmically efficient way to obtain a lower bound is to reduce the number of dates, by imposing further constraints on the latter. We can do this by setting $t_{2k+1} = t_{2k'+1}$ for $2^h \leqslant k, k' < 2^{h+1} - 1$. In this way, the number of different dates is quadratic in the number of nodes, and this induces a complete order on the dates for every h. Therefore, this lower bound can be obtained in polynomial time.

11.3. Bibliographic notes

In this chapter, we have presented a few methods for computing exact worst-case performance bounds. To ease the presentation, we have restricted the approach to tandem networks. All of these methods can also be generalized to tree topologies with no algorithmic cost.

The linear programming approaches can also be generalized to general feed-forward networks for both the arbitrary multiplexing case [BOU 10b, BOU 16c] and the FIFO case [BOU 12b, BOU 16b], but the polynomial algorithms become exponential for two reasons. The first is that the number of constraints depends on the number of paths in the graph ending at server n, and there can be an exponential number of such paths. The second reason is the ordering of the time constraints. In the FIFO framework, the ordering is done the same way as presented in this chapter, i.e. by introducing Boolean variables, but in the arbitrary multiplexing approach, this trick is not possible because it is based on the backlogged periods, and an exponential number of linear programs have to be generated.

Adaptations of the linear program for the arbitrary multiplexing case can be used for other service policies, such as the static priorities in [BOU 11c]. It combines the modular approach to compute constraints on the arrival curves at every flow and every server and incorporate those curves as linear constraints in the linear program.

Other approaches have also been developed for computing worst-case performance bounds in a tandem network. In [BOU 15], a greedy algorithm is developed. It is based on a backward recursion from the last server to the first server, and computes for each flow the maximum amount of backlog that can be transmitted from one server to another. The proof strongly relies on a precise study of the properties of worst-case trajectories.

12

Stability in Networks with Cyclic Dependencies

Until now, we have focused on feed-forward networks, whose underlying graphs are acyclic. In those networks, as soon as the arrival rate to each server is lower than the service rate, the network is stable and worst-case performances can be computed, using, for example, the results of the previous two chapters. The computations can be made by exploring the servers of the networks in a topological order.

When the underlying graph has cycles, it is not possible anymore to analyze the servers in a topological order, and the same techniques cannot be used directly. In fact, computing worst-case performances becomes much more complex, and even the problem of stability becomes difficult and is still open in the network calculus theory.

Very few methods and general results are known for cyclic networks, and this chapter aims to present some of them.

In section 12.1, we define the notion of stability that we call *global stability* to distinguish it from the *local stability* that is defined for each server.

In section 12.2, we describe the most classical technique to obtain sufficient conditions for the stability as the existence of a fix point of an equation. The idea is to decompose the network into acyclic networks, and the fix-point equation relates the performance of each sub-system. This method was first developed by Cruz in [CRU 91b], and then with more detail in [CHA 00, LEB 01].

Section 12.3 focuses on service policies where the global stability is equivalent to the local stability, regardless of the topology of the network. The service policies satisfying this equivalent are service policies that are usually *fair*, in the sense that they

Deterministic Network Calculus: From Theory to Practical Implementation,
First Edition. Anne Bouillard, Marc Boyer and Euriell Le Corronc.
© ISTE Ltd 2018. Published by ISTE Ltd and John Wiley & Sons, Inc.

guarantee a service to flows that need it more. The first focus will be on priorities: two types of priority ensure the stability: when one priority level is defined for each flow (and is different for each flow) and when the flows that have the longest path to go are given the priority (furthest-to-destination-first). The other focus is on the GPS policy, when flow is associated with the same parameter for the whole network.

Finally, in section 12.4, we focus on networks that exhibit instability. As there is, to our knowledge, no simple example of a network that is globally unstable although locally stable, we will focus on examples taken from a slightly different model, the *adversarial model*, that exhibit instability for the FIFO policy and the SDF (shortest-to-destination-first) policy. Note that an example of instability for the FIFO policy in the network calculus framework can be found in [AND 07, AND 00]. Finally, we introduce scaling of cumulative processes in order to exhibit a very simple example of instability in the network calculus framework.

12.1. Network stability

In this section, we define the notion of the *global stability* of a network and that of *local stability*, which is weaker and defines the stability of a server in isolation. The condition for the latter notion is rather intuitive and follows the definition of stability of a queue in queueing theory: the arrival rate must be strictly lower than the service rate. In the context of deterministic network calculus, the notion of average arrival or service rate is not relevant, so we rather deal with long-term arrival or service rate, which we now define.

12.1.1. *Local stability*

DEFINITION 12.1 (Long-term rates).– *Let $\alpha \in \mathcal{C}$ be a sub-additive function and $\beta \in \mathcal{C}$ be a super-additive function.*

1) The long-term arrival rate *of the arrival curve α is*

$$\limsup_{t\to\infty} \frac{\alpha(t)}{t}.$$

2) The long-term service rate *of the service curve β is*

$$\liminf_{t\to\infty} \frac{\beta(t)}{t}.$$

Remark that the long-term arrival rate of a token-bucket arrival curve $\gamma_{r,b}$ is r and the long-term service rate of a rate-latency service curve $\beta_{R,T}$ is R.

In this chapter, we use the notation that has already been defined in section 10.1.1. We also assume strict service curves for all servers. Moreover, we denote by r_i the long-term arrival rate of flow i and by $R^{(h)}$ the long-term service rate for server h.

DEFINITION 12.2 (Local stability).– *A network is said to be locally stable if all its servers are stable using the initial arrival curves, i.e.* $\forall h \in \{1, \dots, n\}$,

$$\sum_{i \in \mathrm{Fl}(h)} r_i < R^{(h)} \quad \textit{or} \quad R^{(h)} = \infty.$$

LEMMA 12.1.– *If server h is locally stable, then*

$$\ell_{\max}\left(\sum_{i \in \mathrm{Fl}(h)} \alpha_i, \beta^{(h)}\right) < \infty,$$

where $\ell_{\max}$ is the maximal length of a backlogged period as defined in Theorem 5.5.

PROOF.– If server h is locally stable, then by definition, $\sum_{i \in \mathrm{Fl}(h)} r_i < R^{(h)}$. Set r'_i for all $i \in \mathrm{Fl}(h)$, and R' such that $r'_i > r_i$ and $\sum_{i \in \mathrm{Fl}(h)} r'_i < R' < R$.

By definition of $\liminf$, there exists t_i such that for all $t \geqslant t_i$, $\alpha_i(t) \leqslant r'_i t$, and as α_i is non-decreasing, we also have for all $t > 0$, $\alpha_i(t) \leqslant \alpha(t_i) + r'_i t$. This with $\alpha_i(0) = 0$ implies that $\alpha_i \leqslant \gamma_{r_i, \alpha_i(t_i)} \overset{def}{=} \gamma_i$.

Similarly, there exists T' such that $\beta(t) \geqslant R'(t - T') = \beta_{R',T'} \overset{def}{=} \beta'$.

To conclude, we use the monotony of $\ell_{\max}$ and the direct computation of the maximum backlogged period for token-bucket and rate-latency curves:

$$\ell_{\max}\left(\sum_{i \in \mathrm{Fl}(h)} \alpha_i, \beta^{(h)}\right) < \ell_{\max}\left(\sum_{i \in \mathrm{Fl}(h)} \gamma_i, \beta'\right) = \frac{\sum_{i \in \mathrm{Fl}(h)} \alpha_i(t_i) + R'T'}{R' - \sum_{i \in \mathrm{Fl}(h)} r'_i} < \infty. \quad \square$$

12.1.2. *Global stability*

DEFINITION 12.3 (Global stability).– *A network is* globally stable *if for each server of the network, the length of its maximal backlogged period is bounded.*

LEMMA 12.2.– *If a network is globally stable, then it is locally stable.*

PROOF.– If the network is globally stable, then the length of the backlogged period is bounded for each server. Let T be the maximum of all of these backlogged periods.

If for each server h, its service curve $\beta^{(h)}$ is replaced by $\beta^{(h)} \vee \delta_T$, the behavior of the network remains unchanged (remember that the service curves are strict, and from Proposition 5.13, only the part of the service curves before T is useful for computing the performances). Therefore, $\beta^{(h)} \vee \delta_T$ is a service curve for server h. But in that case, we have $R^{(h)} = \infty$, so every server is locally stable. □

In the rest of the chapter, and especially when dealing with sufficient conditions for stability, it will be useful to only consider token-bucket and rate-latency curves, which can be deduced from Lemma 12.1, in order to enable this.

LEMMA 12.3.– *Suppose that for a network, when arrival and service curves are, respectively, token-bucket and rate-latency and respect the local stability conditions, the network is also globally stable. Then, the local stability for any arrival and service curve is also a sufficient condition for the global stability.*

PROOF.– Using the proof of Lemma 12.1, if a network is locally stable, then there exist token-bucket arrival curves and rate-latency service curves bounding the original curves and still satisfying the local stability condition. Then, with these arrival and service curves (for which more trajectories are admissible), the network is globally stable, so with the original curves, the network is also globally stable. □

We call the network model that has only token-bucket arrival curves and rate-latency service curves the *linear model.*

Finally, the following lemma will be useful to conclude the proofs in the rest of this chapter.

LEMMA 12.4.– *Let $\alpha_i^{(h)}$ be arrival curves for flow i at the input of server h. If for all h,*

$$\ell_{\max}\left(\sum_{i\in \mathrm{Fl}(h)} \alpha_i^{(h)}, \beta^{(h)}\right) < \infty,$$

then the network is globally stable.

PROOF.– This is a direct consequence of Definition 12.3. □

12.2. The fix-point method: sufficient condition for global stability

This is the most classical method for computing performance bounds in cyclic networks in the network calculus framework. In this section, we slightly generalize the method. Indeed, it is usually presented for rate-latency service curves, token-bucket arrival curves and an elementary network decomposition: arrival constraints are computed for each flow at each server. We present the fix-point method for more general configurations and shapes of arrival and service curves.

In view of Lemma 12.3, we could think it is sufficient to focus on the linear model. However, for several reasons, it is useful to have a more general version:

1) this method can also be generalized to cases where a maximum service curve is known for the servers, and Lemma 12.3 is not valid in that case;

2) the method described below gives not only a sufficient stability condition for the network but also performance bounds, which are improved if the flows and servers can be described by more accurate arrival and service curves.

12.2.1. *Feed-forward transformation*

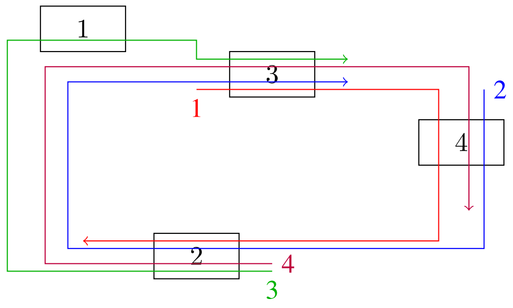
The graph induced by this network is the directed graph $G = ([1, n], \mathbb{A})$, the pair of consecutive servers on a flow. As a result, $G_{\mathcal{N}}$ is a directed and simple graph (with no loop and no multi-edge). A graph can be transformed into an acyclic graph by removing a set of arcs $\mathbb{A}^{\mathrm{r}}$ in its graph. In the network, flows that traverse arcs that are removed are split into several flows.

We now modify the flows in accordance with the arcs that have been removed: each flow i is split into several flows $(i, 1), (i, 2), \dots, (i, m_i)$ of respective paths in $([1, i], \mathbb{A} - \mathbb{A}^{\mathrm{r}})$, $\mathbf{p}_{i,1} = \langle p_i(1), \dots, p_i(k_1)\rangle$, $\mathbf{p}_{i,2} = \langle p_i(k_1 + 1), \dots, p_i(k_2)\rangle$,..., $\mathbf{p}_{i,m_i} = \langle p_i(k_{m_i} + 1), \dots, p_i(\ell_i)\rangle$, where $(p_i(k_j + 1), p_i(k_j)) \in \mathbb{A}^r$. We denote by $\ell_{i,k}$ the length of $\mathbf{p}_{i,k}$ and by $\mathcal{N}^{\mathrm{FF}}$ the feed-forward network that is obtained.

EXAMPLE 12.1.– *The toy network shown in Figure 12.1 (top) can be transformed into an acyclic network by removing arc* $(4, 2)$. *It is also possible to transform it into a tree, with* $\mathbb{A}^{\mathrm{r}} = \{(4, 2), (2, 1)\}$. *With* $\mathbb{A}^{\mathrm{r}} = \mathbb{A}$, *a graph with isolated node only is obtained.*

If $\mathbb{A}^{\mathrm{r}} = \{(4, 2)(2, 1)\}$, *flow* 1 *is split into* $(1, 1)$ *and* $(1, 2)$ *with respective paths* $\mathbf{p}_{1,1} = \langle 3, 4\rangle$ *and* $\mathbf{p}_{1,2} = \langle 2\rangle$, *flow* 2 *is also split into* $\mathbf{p}_{2,1} = \langle 4\rangle$ *and* $\mathbf{p}_{2,2} = \langle 2, 3\rangle$ *and flow* 3 *is split into* $\mathbf{p}_{3,1} = \langle 2\rangle$ *and* $\mathbf{p}_{3,2} = \langle 1, 3\rangle$. *Flow* 4 *remains unchanged* ($\mathbf{p}_{1,4} = \mathbf{p}_4$)*. The result of this decomposition is depicted in Figure 12.1 (bottom).*

Note that the way of separating the flows is not unique and finding the minimum set of edges to remove is an NP-complete problem (it is the *minimum feed-back arc set* problem [GAR 79]). The most classical solution in the literature is to remove every arc – each server is analyzed in isolation – however, it might be a better choice to remove fewer arcs and obtain a tree or a tandem, for example.



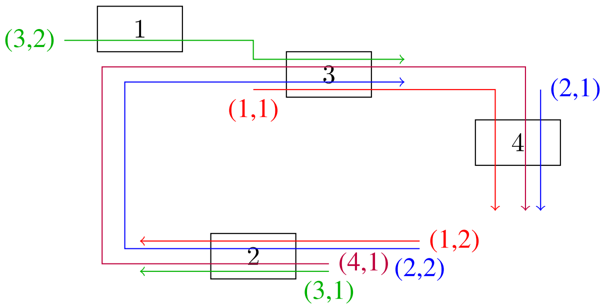


Figure 12.1. *Top: cyclic network of Example 12.1; bottom: its feed-forward transformation. For a color version of this figure, see www.iste.co.uk/bouillard/calculus.zip*

12.2.2. *A fix-point equation*

Consider the network $\mathcal{N}^{\mathrm{FF}}$. If the arrival curve for every flow (i, k) is known and finite, then the performance bounds can be computed for every flow.

In the original network $\mathcal{N}$, the arrival curves of flow i at the first server visited by flow (i, k) in $\mathcal{N}^{\mathrm{FF}}$ are not known for $k > 1$, but we can write equations where these arrival curves are the unknowns. More precisely, we suppose that the network is stable and denote by $\alpha_{i,k}$ the *best* arrival curve (i.e. the minimum arrival curve among all of the possible arrival curves) of flow k at the first server visited by flow (i, k) in $\mathcal{N}^{\mathrm{FF}}$. We note that $\alpha_{i,k}$ is well defined by Proposition 5.2.

There exists a function $F_{i,k}$ such that

$$\alpha_{i,k} \leqslant F_{i,k}((\alpha_{i',k'})_{(i',k')\text{ flow of }\mathcal{N}^{\mathrm{FF}},k>1}).$$

Function $F_{i,k}$ depends on the service curves, the service policies of the servers and the arrival curves of flows $(i, 1)$. The arrival curves (i, k) with $k > 1$ are the variables of the function. Here, $F_{i,k}$ represents any technique or algorithm to compute the arrival curves such as those described in Chapter 10. It can be assumed without loss of generality that $F_{i,k}$ is non-decreasing. Indeed, if $(\alpha_{i',k'})_{(i',k')}$ increases, then more behaviors are possible.

Using a vectorial notation, we can write, with $\boldsymbol{\alpha} = (\alpha_{i,k})_{(i,k) \text{ flow of } \mathcal{N}^{\mathrm{FF}},\ k>1}$ and $\mathbf{F} = (F_{i,k})_{(i,k) \text{ flow of } \mathcal{N}^{\mathrm{FF}},\ k>1}$,

$$\boldsymbol{\alpha} \leqslant \mathbf{F}(\boldsymbol{\alpha}). \qquad [12.1]$$

To summarize, if the network $\mathcal{N}$ is stable, then the equation $\boldsymbol{\alpha} \leqslant \mathbf{F}(\boldsymbol{\alpha})$ has a solution. The following section shows the converse: if there exists a solution to equation [12.1], then network $\mathcal{N}$ is globally stable, and the largest solution of this equation gives arrival curves in the network. The way to prove this result is to introduce stopped times.

12.2.3. *Stopped times*

Consider the network $\mathcal{N}$ where all the arrivals stop at time τ: for all flow i, an arrival curve is then $\alpha_i^\tau : t \mapsto \alpha_i(t \wedge \tau)$. The total amount of data for each flow i and the split flow originating from this flow is bounded by $\alpha_i(\tau)$, and the total amount of data entering the system is finite. As a result, the network is globally stable. Let $(\boldsymbol{\alpha}_{i,k}^\tau)$ be the family of the *best arrival curve* for flow i at the first server visited by flow (i,k). From the previous section

$$\boldsymbol{\alpha}^\tau \leqslant \mathbf{F}^\tau(\boldsymbol{\alpha}^\tau) \leqslant \mathbf{F}(\boldsymbol{\alpha}^\tau),$$

where $\mathbf{F}^\tau$ is obtained by the same way as $\mathbf{F}$, by replacing α_i by α_i^τ. We can assume without loss of generality that $\mathbf{F}^\tau \leqslant \mathbf{F}$ because $\alpha_i^\tau \leqslant \alpha_i$ for each flow i.

We can now prove the main theorem of this section.

THEOREM 12.1.– *Let $\mathcal{C} = \{\boldsymbol{\alpha} \mid \boldsymbol{\alpha} \leqslant \mathbf{F}(\boldsymbol{\alpha})\}$ be the set of solutions of equation* [12.1], *and $\tilde{\boldsymbol{\alpha}} = \sup\{\boldsymbol{\alpha} \mid \boldsymbol{\alpha} \in \mathcal{C}\}$. If $\tilde{\boldsymbol{\alpha}}$ is finite, then $\mathcal{N}$ is globally stable and for all (i,k), $\tilde{\alpha}_{i,k}$ is an arrival curve for flow i at the input of the first server of flow (i,k).*

PROOF.– First, $\tilde{\boldsymbol{\alpha}}$ exists: if $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$ are two elements in $\mathcal{C}$, then for all (i,k), $\alpha_{i,k} \leqslant \mathbf{F}_{i,k}(\boldsymbol{\alpha}) \leqslant \mathbf{F}_{i,k}(\boldsymbol{\alpha} \vee \boldsymbol{\alpha}')$ and similarly, $\alpha'_{i,k} \leqslant F_{i,k}(\boldsymbol{\alpha} \vee \boldsymbol{\alpha}')$, so $\alpha_{i,k} \vee \alpha'_{i,k} \leqslant F_{i,k}(\boldsymbol{\alpha} \vee \boldsymbol{\alpha}')$, and $\boldsymbol{\alpha} \vee \boldsymbol{\alpha}' \in \mathcal{C}$. Then, $\mathcal{C}$ contains a maximum element.

For all $\tau > 0$, $\boldsymbol{\alpha}^\tau \in \mathcal{C}$, so $\tilde{\boldsymbol{\alpha}} \geqslant \boldsymbol{\alpha}^\tau$, and $\tilde{\boldsymbol{\alpha}}$ 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 $\tilde{\boldsymbol{\alpha}}$ is finite. □

In practice, most often, there exists a unique fix-point to the equation $\boldsymbol{\alpha} = \mathbf{F}(\boldsymbol{\alpha})$, so this solution is also the largest solution of $\boldsymbol{\alpha} \leqslant \mathbf{F}(\boldsymbol{\alpha})$.

12.2.4. *Rate-latency service curves and token-bucket arrival curves*

When rate-latency and token-bucket curves are used, $\boldsymbol{\alpha}$ is also a family of token-bucket functions where the arrival rate of flow (i, k) is the same as the one of flow $(i, 1)$. Then, the equation only expresses the bursts, and $\mathbf{F}$ can be written as a system of linear functions: there exist a matrix M and a vector N such that

$$\mathbf{b} = M\mathbf{b} + N,$$

where $\mathbf{b} = (b_{i,k})$ is the family of bursts of flows (i, k). The equation has a solution if and only if the spectral radius of M is less than 1.

Note that if a sufficient condition can be found for the linear model in terms of arrival and service rates, then a similar reasoning to Lemma 12.3 enables us to extend it to arbitrary arrival and service curves with conditions on the long-term arrival and service rates.

12.3. Universally stable service policies

Despite the necessary stability conditions given in the previous section, the general problem of stability remains open in the general case. The previous section focused on the topology and on the characteristics of each flow. In this section, we give stability conditions depending on the service policy.

We assume that all servers have the same service policy and that paths are not cyclic (they do not cross the same server twice). We have seen in Chapter 7 that the performance of a server strongly depends on the service policy, which is also the case for stability. In this section, we prove three policies to be *universally stable*, i.e. whatever the topology of the network, local stability is a sufficient condition for stability. Those three service policies are static priorities, furthest destination first and generalized processor sharing with constant rates.

12.3.1. *Static priority policies*

THEOREM 12.2.– *Consider a network where flows have fixed priorities and a strict priority order on the flows (each flow has a different priority). This network is stable if and only if it is locally stable.*

PROOF.– In view of Lemma 12.3, we only deal here with the linear model. Consider a network with m flows with respective arrival curves $\alpha_i = \gamma_{r_i,b_i}$, and n servers offering respective strict service curves $\beta^{(h)} = \beta_{R^{(h)},T^{(h)}}$. Suppose that flow i has a strictly higher priority than flow j whenever $i < j$.

The condition for local stability translates into $\sum_{i\in\mathrm{Fl}(h)} r_i < R^{(h)}$ for all server h.

We will prove this result by induction on the number of flows in the network. We first define the induction hypothesis H_k.

H_k: the maximum backlog for flows $1,\dots,k$ in each server is finite, and for each server h, the remaining service curve for flows $k+1,\dots,m$ is of the form $\beta^{(h)}_{>k}(t) = R^{(h)}_{>k}[t - T^{(h)}_{>k}]^+$ with

$$R^{(h)}_{>k} > \sum_{i\in\mathrm{Fl}(h)\cap[k+1,m]} r_i \qquad \text{and} \qquad T^{(h)}_{>k} < \infty.$$

H_0 holds: $\beta^{(h)}_{>0} = \beta^{(h)}$.

Suppose now that H_{k-1} holds. Let us consider flow k and note that it has the highest priority among flows $k,\dots,m$. Consider $\mathbf{p}_k$ the path of flow k. We can compute $\alpha_k^{(p_k(i))}$ the arrival curve of flow k after server $p_k(i)$ as $\alpha_k^{(0)} = \alpha_k$ and $\alpha_k^{(p_k(i))} = \alpha_k^{(p_k(i-1))} \oslash \beta^{(p_k(i))}_{>k-1}$, with the convention $p_k(0) = 0$. From the induction hypothesis H_{k-1}, $R^{(h)}_{k-1} \geqslant r_k$, so $\alpha_k^{(p_k(i))}$ is a token bucket equal to $\gamma_{r_k, b_k^{(p_k(i))}}$ with $b_k^{(p_k(i))} = b_k^{(p_k(i-1))} + r_k T^{(p_k(i))}_{>k-1} < \infty$. The maximum backlog for flow k at server $p_k(i)$ is b_k, so it is finite, and from Theorem 7.6, the residual strict service curve offered to flows $k+1,\dots,m$ by server $h = p_k(i)$ is $\beta^{(h)}_{>k} = \left[\beta^{(h)}_{>k-1} - \alpha_k^{(h)}\right]^+$. Then, $\beta^{(h)}_{>k}$ is rate-latency with rate $R^{(h)}_{>k-1} - r_k > \sum_{i\in\mathrm{Fl}(h)\cap[k+1,m]} r_i$ from the induction hypothesis. The service curves of the servers not crossed by flow k are not modified. Then, H_k holds.

Finally, H_m holds and this is exactly the global stability condition: we proved that for every flow i and server h it crosses, there exists an arrival curve $\gamma_{r_i,b_i^{(h)}}$ for this flow before server h, and $\sum_{i\in\mathrm{Fl}(h)} r_i < R^{(h)}$, so the maximum backlogged period for each server h is bounded. □

12.3.2. *Furthest destination first*

The furthest destination first (FDF) policy is also based on the static priorities. Here, priorities might be set differently for each server: they depend on the number of servers yet to be crossed by the flow, and the priority is given to flows that are the furthest from their destination. Flows at equal distance from their destination have an arbitrary policy between them.

EXAMPLE 12.2.– *Consider again the network shown in Figure 12.1. In server 3, flow 1 has the highest priority, followed by flow 4. Flows 2 and 3 share the same priority. In server 2, flows 4 and 3 share the highest priority, and flow 1 is given the lowest priority.*

THEOREM 12.3.– *The local stability condition is a sufficient condition for global stability under the FDF policy.*

PROOF.– We use the same notations as in section 10.1.1 for the network, and from Lemma 12.3, we focus on the linear model.

We proceed by induction on the total length of the flows. Note that the length of flow i is denoted by ℓ_i. We denote by $L = \sum_{i=1}^{m} \ell_i$ the total length of the flows.

H_ℓ: every network that is locally stable and for which the total length of the flows is at most ℓ is stable.

H_0 is satisfied: there is no flow in the network.

Suppose that $H_{\ell'}$ is satisfied for all $\ell' < \ell$ and consider a network with a total length of ℓ. Consider any flow i of maximal length and let h be the first server it crosses. In this server, this flow has the highest priority. This highest priority is possibly shared by other flows in this server. These flows then have the same length as flow i, and server h is also the first server they cross: otherwise, it would refute the maximality of the length of flow i.

Set J the set of these flows (including i). We modify the network as follows: we replace server h by $|J| + 1$ servers, (h, j) for $j \in J$ and $(h, 0)$. Each server (h, j), $j \in J$ is crossed by flow j only and offers a min-plus service curve $\beta^{(h,j)} = \left[\beta^{(h)} - \sum_{k\in J,k\neq j} \alpha_k\right]_\uparrow^+$, and server $(h, 0)$ is crossed by the flows in $\mathrm{Fl}(h)\backslash J$ and offers a strict service curve $\beta^{(h,0)} = \left[\beta^{(h)} - \sum_{k\in J} \alpha_k\right]_\uparrow^+$.

If this modified network is globally stable, so is the original network: flow $j \in J$ has the highest priority, so it is guaranteed a service curve $\beta^{(h,j)}$ (flow j is in competition only with the other flows of J), and from Theorem 7.6, the lower-priority flows are guaranteed a strict service curve $\beta^{(h,0)}$. It is easy to check that this new network is also locally stable.

We now transform again the network in order to apply the induction hypothesis: remove server (h, j) for all flows $j \in J$ and replace the arrival curve (at the second server crossed by flow j) by $\alpha_j \oslash \beta^{(h,j)}$. As server h is locally stable, these curves are a token bucket with rate r_j. The new network obtained is locally stable, and the sum of the lengths of the flows is now $\ell - |J|$, so by $H_{\ell-|J|}$, it is stable.

It is clear that if this new network is stable, then the original network is also stable. This proves that H_ℓ is satisfied and H_L also holds. □

12.3.3. *General processor sharing policies with constant rates*

We now show the universal stability of an important class of GPS policy: there is only one parameter for each flow that does not change on the path followed by the flow. Then, each flow i is associated with a non-negative number ϕ_i, and is guaranteed a proportion of $\phi_i/(\sum_{j\in\mathrm{Fl}(h)} \phi_j)$ of the service in server h. We call this type of network *GPS with fixed parameters*.

Here again, we use the linear model and each flow i has arrival curve $\alpha_i = \gamma_{r_i,b_i}$ and each server h offers a strict service curve $\beta^{(h)} = \beta_{R^{(h)},T^{(h)}}$.

LEMMA 12.5.– *With the notations defined above, there exists a flow i such that for every server $h \in \mathbf{p}(i)$,*

$$r_i < \phi_i \frac{R^{(h)}}{\sum_{j\in\mathrm{Fl}(h)} \phi_j}.$$

PROOF.– Consider a flow $i = \mathrm{Argmax}_{j=1}^{m} \phi_j/r_j$, i.e. for all $j \in \{1,\ldots,m\}$, $\phi_i/r_i \geqslant \phi_j/r_j$. We now proceed by contradiction and assume that $r_i \geqslant \phi_i \frac{R^{(h)}}{\sum_{j\in\mathrm{Fl}(h)} \phi_j}$. For every other flow k, we would also have $r_k \geqslant \phi_k r_i/\phi_i \geqslant \phi_k R^{(h)}/(\sum_{j\in\mathrm{Fl}(h)} \phi_j)$. But $\sum_{k\in\mathrm{Fl}(h)} r_k \geqslant (\sum_{k\in\mathrm{Fl}(h)} \phi_k) R^{(h)}/(\sum_{j\in\mathrm{Fl}(h)} \phi_h) = R^{(h)}$, and the network would not be locally stable. □

THEOREM 12.4.– *The local stability is a sufficient condition for the global stability.*

PROOF.– As in the two previous theorems, we restrict the analysis to the linear model without loss of generality and proceed by induction with hypothesis.

H_k: for any GPS network with k flows with fixed parameters, the local stability condition is a global stability condition.

H_0 naturally holds as there is no flow in the network.

Suppose that H_{k-1} holds for $k > 0$ and consider a network with k flows that is locally stable. Let $i \in \mathrm{Argmax}_{j=1}^{m} \phi_i/r_i$. From Lemma 12.5,

$r_i < \phi_i R^{(h)}/(\sum_{j\in\mathrm{Fl}(h)} \phi_j)$, so the service curve $\beta_i^{(h)} = \frac{\phi_i}{\sum_{j\in\mathrm{Fl}(h)} \phi_j}\beta^{(h)}$ offered to flow i is enough to bound the arrival curve of flow i at each of its server: there, $\alpha_i^{(p_i(\ell))} \leqslant \alpha_i^{(p(\ell-1))} \oslash \beta_i(h)$, and there exists $b_i^{(h)}$ such that $\alpha_i^{(h)} = \gamma_{r_i, b_i^{(h)}}$.

But from Lemma 7.4, the servers of $\mathbf{p}_i$ offer to the other flows a strict service curve $\tilde{\beta}^{(h)} = \left[\beta^{(h)} - \alpha_i^{(h)}\right]^+$ and there exists T such that $\tilde{\beta}^{(h)} = \beta_{R^{(h)}-r_i,T}$.

When removing flow i, we then obtain a locally stable network with $k-1$ flows, which, from H_{k-1}, is globally stable. Therefore, the original network is stable and H_k holds. □

Note that constant rates include GPS with proportional rates ($\phi_i = r_i$ for all i). In this case, the stability is also easily obtained as the residual service curve is sufficient to ensure the local stability for every flow independently of the others. More detailed results on GPS can be found in [PAR 94].

12.4. Instability of some systems

12.4.1. *Adversarial methods*

The adversarial method is an alternative method for studying the stability of a network. Rather than studying a given network described by its servers, their service policy and the flows circulating among these servers, it aims to answer stability issues given a topology (servers and their connections). An example question would be: *is there a protocol that makes the network unstable?*

The main idea of the approach is to define an *adversary* that will inject data into the network. The adversary can choose where to inject data and the path followed by these. The constraints it has to respect are the network topology (the paths followed by the data is a path of the network topology) and that the network is locally stable (servers serve data at a rate of exactly 1 and the global injection rate in each server is less than 1).

We show that we can construct an unstable network based on the topology presented in Figure 12.2 for two different service policies: the shortest-to-destination first (SDF) and the FIFO policies.

The following are some differences with network calculus:

1) on the one hand, the service rate is constant and equal to 1. In this sense, the results are stronger: if the adversarial method leads to instability, then models with more general service guarantees will also lead to instability;

2) on the other hand, with the adversarial method, the arrivals are not constrained by functions α. It is a stronger condition than the local stability, and rather a local stability for the whole arrival process. For example, to show the instability of a network, data are injected at rate $0.5 < r < 1$, but this rate is shared by two flows: there are intervals of time when data are only sent by flow 1 and data are sent by flow 2 outside of these intervals. These intervals can be arbitrarily long, so the arrival curve would have a rate r for each curve and the total arrival rate would be $2r > 1$ in the network calculus framework. In that sense, the adversarial creates more instability than network calculus.

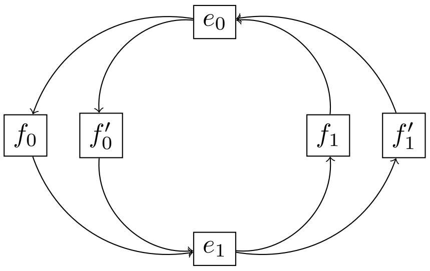


Figure 12.2. *Unstable topology for the adversarial method*

As a consequence, instability results that will be described cannot be applied to the network calculus framework, although the models exhibit some similarities. Nonetheless, they give some hints about what is a stable policy. For example, the universal stability of FDS was first set in that setting.

12.4.1.1. *Instability of the shortest-to-destination first policy*

The shortest-to-destination first (SDF) policy gives priority to flows that have the shortest path to go to reach their destination. Ties are broken arbitrarily. This is a policy that is known to be the *worst* in tandem networks, in the sense that it will lead to the largest performance bounds [BOU 11c]. It is then natural to think that this policy leads to instability.

Consider the example of Figure 12.2. Each server serves data at a rate of exactly 1, and data are injected in the network at rate r. We also assume that there is one unit of data in server e_0 at time 0. The path that follows those data is $\langle e_0 \rangle$. We will exhibit a sequence of times for which the amount of data in the network increases. Time is divided into phases, and we look at the amount of data at the end of each phase.

We now precisely describe one phase, assuming that there are s (initially $s = 1$) units of data in server e_0 that exit the network at this server (hence having the highest priority):

1) during unit of time s, data are injected at rate r into the path $\langle e_0, f_0, e_1\rangle$. As the data initially in the network have the priority, these data (rs units) will be served from time s to time $s + rs$;

2) during unit of time rs, data are injected at rate r along the path $\langle e_0, f'_0, e_1\rangle$ in the network, producing r^2s units of data, served from time $s+rs$ to time $s+rs+r^2s$;

3) during the same period of time (from time s to time $s + rs$), packets are also injected along the path $\langle f_0\rangle$ at rate r. This flow and the data that arrives between times 0 and s interfere. This flow has the priority, and at time $s+rs$, there are $(1+r)rs-rs = r^2s$ units of data in server f_0.

Flows of this phase are depicted in Figure 12.3, and Figure 12.4 illustrates the processes in servers e_0, f_0 and e_1 during the first phase.

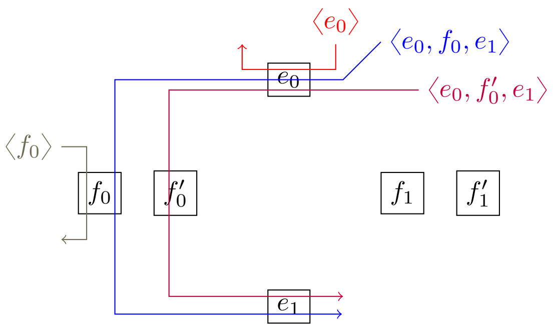


Figure 12.3. *Flow description for one phase of the instability in SDF. For a color version of this figure, see www.iste.co.uk/bouillard/calculus.zip*

At time $s+rs$, there are $2r^2s$ units of data in the network. These data arrive at rate 2 in server e_1. For the next phase, we exchange the roles of 0 and 1 in the numbering of the servers. These data will have the priority over data that would be injected in paths $\langle e_1, f_1, e_0\rangle$ and $\langle e_1, f'_1, e_0\rangle$. Therefore, the configuration is exactly the same, with $2r^2s$ units of data. Thus, if $2r^2 > 1$, the amount of data strictly increases, which makes the network unstable as soon as $r > 1/\sqrt{2}$.

12.4.1.2. *Instability of the FIFO policy*

The example is nearly the same as that for the SDF policy. The behavior is different from the third step of the phase:

1) during the same period (from time s to time $s + rs$), packets are also injected along the path $\langle f_0 \rangle$ at rate r. This flow and the data arrived between times 0 and s interfere. The service policy is FIFO, so data are served respectively at rate $1/(1 + r)$ and $r/(1 + r)$. At time $s + rs$, it remains $rs(1 - \frac{1}{1+r}) = \frac{r^2 s}{1+r}$ units of data in the flow continuing to server e_1;

2) between time $s(1+r)$ and time $s(1+r+r^2)$, r^3 units of data are injected to the system, following path $\langle e_1 \rangle$. During this period, the total amount of data that arrives is then $r^2 s + \frac{r^2 s}{1+r} + r^3 s$, and the amount of data that is served is $r^2 s$. Therefore, it remains $\frac{r^2 s}{1+r} + r^3 s$ units of data at time $s(1 + r + r^2)$, which is the end of the phase.

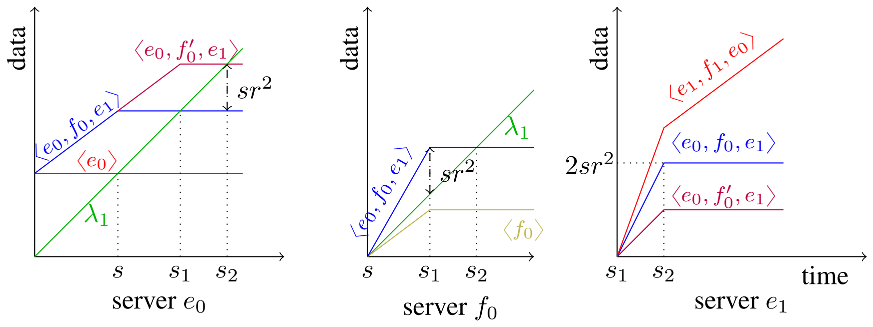


Figure 12.4. *The SDF policy is unstable in the adversarial model, with $s_1 = s + sr$ and $s_2 = s + rs + r^2 s$. The amounts of data are represented in a cumulative way. For instance, $\langle e_0, f_0, e_1 \rangle$ represents the cumulative arrivals of $\langle e_0 \rangle + \langle e_0, f_0, e_1 \rangle$. For a color version of this figure, see www.iste.co.uk/bouillard/calculus.zip*

The next phase is similar up to inverting the roles of 0 and 1 in the numbering of the servers. Therefore, if $\frac{r^2}{1+r} + r^3 > 1$, the amount of data in the network grows at each phase and the network is unstable. This is the case for $r > 0.85$.

12.4.2. *Instability in network calculus with scaling*

In this section, we make a small extension to network calculus, by allowing the *scaling* of data. Then, we present a very small network that can be unstable although it is locally stable. This example was first given in [CHA 00].

DEFINITION 12.4 (Scaled flow).– *Let $A \in \mathcal{C}$ be a cumulative process of a flow and $m \in \mathbb{R}^+$. The scaled flow with factor m has cumulative process mA.*

LEMMA 12.6.– *If A is α-upper constrained, then mA is $m\alpha$ upper-constrained.*

PROOF.– $\forall s, t \geqslant 0$, with $s \leqslant t$, $mA(t) - mA(s) = m(A(t) - A(s)) \leqslant m\alpha(t-s)$. □

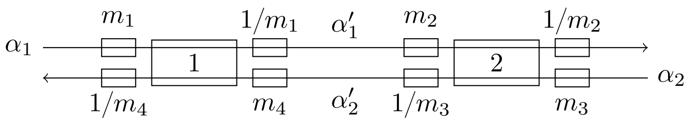


Figure 12.5. *Unstable network*

Scaling enables us to deal with more general service schemes. For example, in the network of Figure 12.5, data of flow 1 are first scaled by a factor m_1 when it crosses server 1. Then, it is scaled by $1/m_1$, in order to be back in its original scale. The same holds in the second server with scaling factor m_2, and for flow 2 with scaling factors m_3 in server 2 and m_4 in server 1. If both servers serve exactly one bit of data per unit of time, and $m_1 < m_4$ and $m_3 < m_2$, one unit of data of flow 2 takes longer to process than one unit of data of flow 1 in server 1, but shorter in server 1. Consider the network shown in Figure 12.5 with $\beta^{(1)} = \beta^{(2)} = \lambda_1$, $\alpha_1 = \alpha_2 = \gamma_{1,1}$. First, the network is locally stable if $m_1 + m_4 < 1$ and $m_2 + m_3 < 1$, which we will assume from now on.

Let us first apply the fix-point method.

Let us denote by α'_1 and α'_2 the arrival curves of flow 1 and flow 2, respectively, between the two servers computed with Corollary 5.3, Theorem 7.1 and Lemma 12.6. We have

$$\alpha'_1 = \frac{1}{m_1}\left(m_1\alpha_1 \oslash \left[\beta^{(1)} - m_4\alpha'_2\right]_{\uparrow}^{+}\right) \qquad \text{and}$$

$$\alpha'_2 = \frac{1}{m_3}\left(m_3\alpha_2 \oslash \left[\beta^{(2)} - m_2\alpha'_1\right]_{\uparrow}^{+}\right).$$

Denoting $\alpha'_i = \gamma_{1,\sigma_i}$, we have

$$\sigma'_1 = 1 + \frac{m_4\sigma'_2}{1 - m_4} \qquad \text{and} \qquad \sigma'_2 = 1 + \frac{m_2\sigma'_1}{1 - m_2}.$$

Replacing σ'_2 by its expression in the first equation leads to

$$\sigma'_1 = 1 + \frac{m_4}{1-m_4} + \frac{m_2 m_4}{(1-m_2)(1-m_4)}\sigma'_1,$$

and we find a finite solution if $\frac{m_2 m_4}{(1-m_2)(1-m_4)} < 1$, i.e. if $m_2 + m_4 < 1$, which is a stronger condition than the local stability.

Now let us prove that if $m_2 + m_4 > 1$, then the network is not globally stable. For this aim, we build a trajectory where the backlog can grow arbitrarily large. We assume that server 1 gives higher priority to flow 2 and server 2 gives higher priority to flow 1.

The arrival processes are exactly $A_1 = \gamma_{1,0}$ and $A_2 = \gamma_{1,1}$.

Contrary to the adversarial method model, here, we only assume minimal service, and then allow instantaneous transmission of data between servers 1 and 2:

1) at time 0+, a burst of size 1 arrives for flow 2 at server 1 (the service is infinite at this time at server 2), after scaling the burst is of size m_4;

2) until time $\frac{m_4}{1-m_4}$, server 1 serves flow 2 only, and the backlog of flow 1 in server 1 is then $x = \frac{m_1 m_4}{1-m_4}$;

3) at time $t_1 = \frac{m_4}{1-m_4}$, the backlog of flow 1 is instantaneously served and enters server 2. Due to the scaling, the quantity of data to enter is $\frac{m_2}{m_1}x = \frac{m_2 m_4}{1-m_4}$;

4) flow 1 is served with higher priority and takes time $\frac{m_2}{(1-m_4)(1-m_2)}$ to empty. The backlog for flow 2 is then $y = m_3\frac{m_2}{(1-m_4)(1-m_2)}$;

5) at time $t_2 = t_1 + \frac{m_2}{(1-m_4)(1-m_2)}$, this quantity of data is transferred to server 1. The scaling quantity of data in server 1 is $\frac{m_3 y}{m_3} = \frac{m_4 m_2}{(1-m_4)(1-m_2)}$.

When $m_2 + m_4 > 1$, $\frac{m_4 m_2}{(1-m_4)(1-m_2)} > 1$ and the quantity of data in server 1 at step 5 has strictly increased compared to step 1 in server 1.

The process continues the same way: when there is no backlog left of the higher-priority flow, the flow with lower priority is instantaneously transferred to the other server, where it has higher priority. The sequence of the quantity of data transferred from server 2 to server 1 strictly increases at each round, so the system is unstable. The evolution of the backlogs that are transferred is depicted in Figure 12.6.

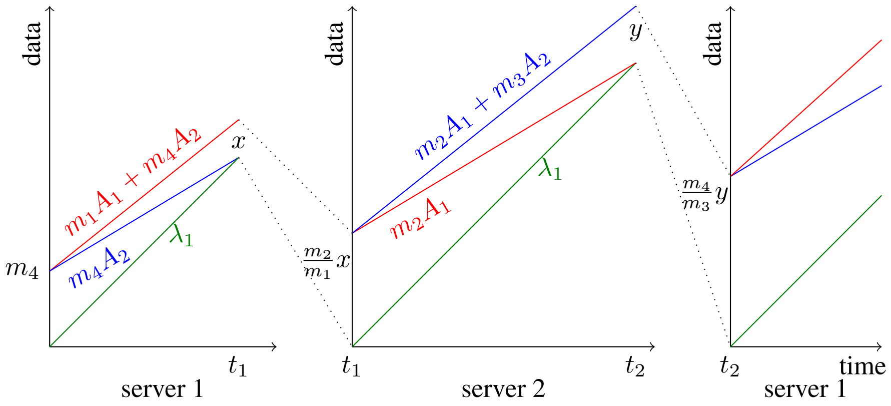


Figure 12.6. *Evolution of the backlogs in the unstable network. For a color version of this figure, see www.iste.co.uk/bouillard/calculus.zip*

12.5. Bibliographic notes

The theoretical aspects of the stability in deterministic queueing networks have mainly been studied in the adversarial queueing network model that we briefly described in section 12.4.1. This model can be described in terms of one *injection rate* instead of one arrival curve per flow and one service curve per server. Note that the injection rate is the rate at which data are sent into the network, and a service policy is universally stable if for all injection rates less than 1, the network is stable. A precise presentation of the model can be found in [BOR 01]. Results on the universal stability can be found in [AND 01, ÀLV 04] for FIFO, LIFO and various priority policies. In the latter reference, the authors also show that the universal stability can be formalized into a minor-exclusion problem. In [LOT 04], the authors show sufficient and necessary conditions for the stability, depending on the length of the flows that are injected. They show that if d is the maximal length of a flow and the injection rate is less than $1/(d+1)$, then the network is stable for any service policy. This sufficient condition also holds in the network calculus model.

Fewer works concern the stability in the network calculus framework. The most classical result is the stability of the ring, which is proved in [TAS 96] for work-conserving links and generalized in [LEB 01]. Instability results are proved in [AND 07, AND 00]. In [AND 07], the authors even show that the FIFO policy can be unstable at arbitrarily small rates. The construction of an unstable network is too complex to be provided here. Some works have focused on finding sufficient condition for the stability in FIFO networks, among which we can cite [RIZ 08].

Concerning the GPS policy, the results presented in this chapter come from the work by Parekh and Gallagher in [PAR 94]. This direction of research has been continued by Barta and Nemeth *et al.* in [BAR 01, NEM 05]. The authors study more general choices of parameters and present an algorithm in order to give a sufficient condition for the stability based on the fix-point method.

Another direction of research consists in transforming the network to ensure global stability. One method proposed consists of breaking the cyclic dependencies. Removing arcs could disconnect the network, but forbidding some paths of length 2, as in the *turn-prohibition* method [STA 02, PEL 04], can ensure both stability and connectivity.