DEBORAH: A Tool for Worst-Case Analysis of FIFO Tandems

Luca Bisti, Luciano Lenzini, Enzo Mingozzi, and Giovanni Stea

Dipartimento di Ingegneria dell'Informazione, University of Pisa
Via Diotisalvi 2, I-56122 Pisa, Italy
{luca.bisti,l.lenzini,e.mingozzi,g.stea}@iet.unipi.it

Abstract. Recent results on Network Calculus applied to FIFO networks show that, in the general case, end-to-end delay bounds cannot be computed in a closed form, whereas solving non-linear programming problems is instead required. Furthermore, it has been shown that these bounds may be larger than the actual worst-case delay, which also calls for computing *lower* bounds on the latter. This paper presents DEBORAH (Delay Bound Rating AlgoritHm), a tool for computing upper and lower bounds to the worst-case delay in FIFO tandem networks. DEBORAH can analyze tandems of up to several tens of nodes in reasonable time on off-the-shelf hardware. We overview the various algorithms used by DEBORAH to perform the various steps of the computations, and describe its usage.

1 Introduction

Computing *good* end-to-end delay bounds in networks employing per-aggregate resource management (such as DiffServ [2] or MPLS [4]) is a challenging task. A delay bound is as good as it is *tight*, i.e. close to the actual *worst-case delay* (WCD) for a bit of the flow being analyzed. While it is fairly simple to compute the WCD under per-flow resource management [3], this task appears to be considerably more complex in networks employing per-aggregate resource management. A recent work shows that WCD computation is actually NP-hard under *blind* multiplexing [15], i.e. where no assumption is made regarding the flow multiplexing criterion (e.g. both FIFO and strict-priority fit this definition). Tools for computing delay bounds in the above settings have also been released [12-13,15]. As for FIFO-multiplexing networks (where the FIFO hypothesis generally allows for smaller delays), our previous works [1,9-11] describe a methodology for computing per-flow delay bounds in tandem networks of rate-latency nodes traversed by leaky-bucket shaped flows. The method, called *Least Upper Delay Bound* (LUDB), is based on the well-known Network Calculus theorem that allows a parametric set of *per-flow* service curves to be inferred from a *per-aggregate* service curve at a single node. As shown in [11], we can derive end-to-end service curves only for *nested* tandems, where the path traversed by a flow *a* is either entirely included into the path of another flow *b* or has a null intersection with it. Non-nested tandems, instead, have to be *cut* into a number of nested sub-tandems, which have to be analyzed separately using LUDB. Then, *per sub-tandem* delay bounds are computed and summed up to obtain the end-to-end delay bound. In this case, there are always several ways for cutting a tandem, and there is no way to state *a priori* whether one leads to better results than the others. The algorithmic difficulties involved into computing the LUDB are non trivial: closed-form solutions are known only for specific topologies (e.g., tree networks [10] and tandems with 1-hop persistent interfering traffic [9]). In the other cases, the piecewise linear problem that comes out of (sub-)tandem analysis has not been proved to be convex. Moreover, the number of possible cuts for a non-nested tandem may grow exponentially with the tandem length. Therefore, software tools that allow tandem analysis are necessary even for small-scale problems. Moreover, recent research [1] has shown that the LUDB may actually be larger than the WCD. Therefore, it is also important to compute *lower bounds* on the WCD, which requires the ability to perform operations (e.g., multiplexing/demultiplexing and convolution) on cumulative arrival functions.

In this paper we present the algorithms used within DEBORAH (DElay BOund Rating AlgoritHm, [14]), an open-source C++ tool, for computing upper and lower bounds on the WCD. The algorithms are in general exponential, but – thanks to clever implementations – appears to be fast enough to manage tandems of up to several tens of nodes within a reasonable time.

The rest of the paper is organized as follows: we report some background on Network Calculus in Section 2. Section 3 reports the system model. We describe the LUDB methodology in Section 4, and Section 5 highlights the algorithms coded into DEBORAH and presents its interface. Section 6 concludes the paper.

2 Network Calculus Background

We report a minimal background on Network Calculus, essentially with the purpose of laying down the notation for the remainder. Interested readers are referred to the abundant literature for further information [3,5-6,8,16].

Let $A(t)$ and $D(t)$ be the (wide-sense increasing) *Cumulative Arrival* and *Cumulative Departure* functions (CAF, CDF) for a flow at a network element. That network element can be modeled by the *service curve* $\beta(t)$ if:

$$D(t) \geq \inf_{0\leq s\leq t}\left\{A(t-s)+\beta(s)\right\} = (A\otimes\beta)(t), \tag{1}$$

for any $t \geq 0$. The flow is said to be guaranteed the (minimum) service curve β, which is the min-plus convolution ($\otimes$) of A and β. Min-plus convolution is commutative and associative, and convolution of concave curves with $f(0)=0$ is equal to their minimum. Several network elements, such as delay elements, links, and regulators, can be modeled by service curves. For example, network elements which have a transit delay bounded by φ can be described by the following service curve:

$$\delta_\varphi(t) = \begin{cases} +\infty & t \geq \varphi \\ 0 & t < \varphi \end{cases}.$$

Many commonplace packet schedulers can be modeled by a family of simple service curves called the rate-latency service curves, defined as $\beta_{\theta,R}(t) = R \cdot [t-\theta]^+$ for some $\theta \ge 0$ (the latency) and $R \ge 0$ (the rate). Notation $[x]^+$ denotes $\max\{0,x\}$. A fundamental result of Network Calculus is that the service curve of a feed-forward sequence of network elements traversed by a data flow is obtained by convolving the service curves of each of the network elements.

A*rrival curves* constrain the CAFs. A wide-sense increasing function α is said to be an arrival curve for CAF A if $A(t) - A(\tau) \le \alpha(t-\tau)$, for all $\tau \le t$. A flow regulated by a *leaky-bucket* shaper, with *sustainable rate* ρ and *burst size* σ, is constrained by the *affine* arrival curve $\gamma_{\sigma,\rho}(t) = (\sigma + \rho \cdot t) \cdot 1_{\{t>0\}}$. Function $1_{\{expr\}}$ is equal to 1 if *expr* is true, and 0 otherwise.

End-to-end delay bounds can be derived by combining together arrival and service curves. Given a FIFO network element (or tandem thereof) characterized by a service curve β and a flow traversing it, constrained by arrival curve α, the delay is bounded by the horizontal deviation:

$$h(\alpha,\beta) \triangleq \sup_{t\ge 0}\left[\inf\left\{d \ge 0 : \alpha(t-d) \le \beta(t)\right\}\right]. \tag{2}$$

From (2) it follows that $\beta_1 \le \beta_2 \Rightarrow h(\alpha,\beta_1) \ge h(\alpha,\beta_2)$. Notation $\beta_1 \le \beta_2$ means that $\forall t \;\; \beta_1(t) \le \beta_2(t)$. Regarding FIFO multiplexing of flows into the same service curve element, a fundamental result ([7], [3], Chapter 6) allows one to derive *per-flow* service curves from *per-aggregate* ones:

Theorem 1 (FIFO Minimum Service Curves, [3])
Consider a lossless node serving two flows, 1 and 2, in FIFO order. Assume that packet arrivals are instantaneous. Assume that the node guarantees a minimum service curve β *to the aggregate of the two flows. Assume that flow 2 has* α_2 *as an arrival curve. Define the family of functions:*

$$E(\beta,\alpha,\tau)(t) = \left[\beta(t) - \alpha_2(t-\tau)\right]^+ \cdot 1_{\{t>\tau\}} \tag{3}$$

For any $\tau \ge 0$ *such that* $E(\beta,\alpha,\tau)(t)$ *is wide-sense increasing, then flow 1 is guaranteed the (equivalent) service curve* $E(\beta,\alpha,\tau)(t)$.

Theorem 1 describes an *infinity* of equivalent service curves, each instance of which (obtained by selecting a specific value for the τ parameter), is a service curve for flow 1, provided it is wide-sense increasing. Hereafter, we omit repeating that curves are functions of time (and, possibly, of other parameters such as τ) whenever doing so does not generate ambiguity.

It has been proved in [10-11] (to which the interested reader is referred for more details and proofs) that *pseudoaffine* curves effectively describe the service received by single flows in FIFO multiplexing rate-latency nodes. We call a pseudoaffine curve one which can be described as:

$$\pi = \delta_D \otimes \left[\bigotimes_{1\le x\le n} \gamma_{\sigma_x,\rho_x} \right] = \delta_D \otimes \left[\bigwedge_{1\le x\le n} \gamma_{\sigma_x,\rho_x} \right], \tag{4}$$

i.e., as a multiple affine curve shifted to the right, the latter inequality stemming from the fact that affine curves are concave. We denote as *offset* the non-negative term D, and as *leaky-bucket stages* the affine curves between square brackets. We denote with ρ_π^* (*long-term rate*) the smallest sustainable rate among the leaky-bucket stages belonging to the pseudoaffine curve π, i.e. $\rho_\pi^* = \min_{x=1,\dots,n}(\rho_x)$. A rate-latency service curve is in fact pseudoaffine, since it can be expressed as $\beta_{\theta,R} = \delta_\theta \otimes \gamma_{0,R}$. Although more general than rate-latency curves, pseudoaffine curves are still fairly easy to manage from a computational standpoint: the convolution of two pseudoaffine curves is in fact still a pseudoaffine curve. Furthermore, Theorem 1 can be specialized for the case of pseudoaffine service curves and leaky-bucket arrival curves as follows:

Corollary 2 [10]
Let π be a pseudoaffine service curve, with offset D and n leaky-bucket stages γ_{σ_x,ρ_x}, $1 \le x \le n$, and let $\alpha = \gamma_{\sigma,\rho}$, with $\rho_\pi^* \ge \rho$. If a node guarantees a minimum service curve π to the aggregate of the two flows, and flow 2 has α as an arrival curve, then the family of functions $\left\{\overline{E}(\pi,\alpha,s),\, s \ge 0\right\}$, with:

$$\overline{E}(\pi,\alpha,s) = \delta_{h(\alpha,\pi)+s} \otimes \left[\bigotimes_{1\le x\le n} \gamma_{\rho_x\{s+h(\alpha,\pi)-D\}-(\sigma-\sigma_x),\rho_x-\rho} \right], \tag{5}$$

are pseudoaffine equivalent service curves for flow 1. Furthermore, set $S \triangleq \left\{\overline{E}(\pi,\alpha,s),\, s \ge 0\right\}$ includes all the equivalent service curves which are relevant for computing delay bounds.

Therefore, all the “good” performance bounds that can be found by applying Theorem 1 can also be found by applying Corollary 2. As (5) is much more compact than (3), and perfectly equivalent from the point of view of computing performance bounds, we will use the former henceforth.

3 System Model

We analyze a tandem of N *nodes*, connected by links and traversed by flows, i.e. distinguishable streams of traffic. We are interested in computing a tight end-to-end delay bound for a *tagged flow*, which traverses the whole tandem from node 1 to N. At each node, *FIFO multiplexing* is in place, meaning that all flows traversing the node are buffered in a single queue First-Come-First-Served. Furthermore, the aggregate of the flows traversing a node is guaranteed a *rate-latency* service curve, with rate R^k and latency θ^k, $1 \le k \le N$. A flow can be identified by the couple (i, j), $1 \le i \le j \le N$, where i and j are the first and last node of the tandem at which the flow is multiplexed with the aggregate. Flows are *fluid*, i.e. we assume that it is feasible to inject and service an arbitrarily small amount of traffic at a node, and are constrained by a σ, ρ *leaky-bucket* arrival curve at their ingress node. Leaky-bucket curves are additive, hence we can safely assume that at most *one* flow exists along a path and identify it using the path as a subscript. Based on how the paths of their flows are interleaved, tandems can be either *nested* or *non-nested*. In a *nested* tandem, flows are either *nested* into one another, or they have null intersection, i.e. no two flows (i,j), (h,k) exist with $i<h\le j<k$. Fig. 1a represents a nested tandem of three nodes, where flow $(3,3)$ is nested within flow $(2,3)$. Furthermore, flows $(1,1)$, $(3,3)$ and $(2,3)$ are nested within the tagged flow $(1,3)$. The *level of nesting* $l(i,j)$ is the number of flows (h,k) into which (i,j) is nested. With reference to Fig. 1a, it is $l(1,1)=l(2,3)=2$, $l(3,3)=3$, and $l(1,3)=1$. The *level of nesting of the tandem* is the maximum level of nesting of one of its flows, i.e. the maximum number of flows crossing a single node. A tandem of N nodes has a level of nesting no greater than N, and that the maximum number of distinguishable flows in an N-node nested tandem is $2N-1$.

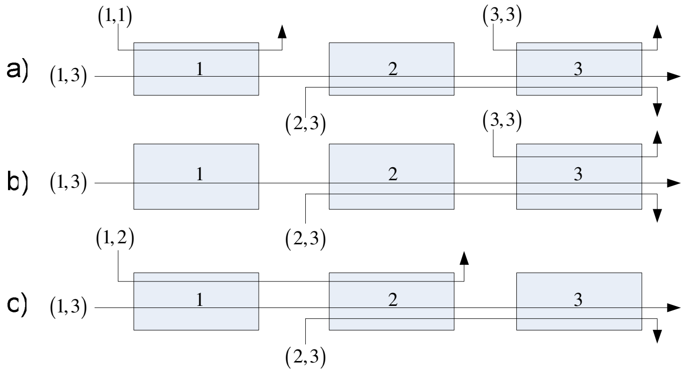


Fig. 1. a) A nested tandem; b) a fully nested tandem; c) a non-nested tandem

A particular case of n-level nested tandem is the one where there is only *one* flow at each level of nesting, i.e. $\forall x, 1\le x\le n, \exists!(i,j): l(i,j)=x$. We call such a tandem a *fully nested tandem*. A sink-tree tandem, where there are exactly N flows (i,N), $1\le i\le N$ (see Fig. 1b, above), is a fully nested tandem (whose level of nesting is N). On the other hand, a tandem is non-nested if it does not verify the above definition, as the one shown in Fig. 1c. In that case, we say that flow $(1,2)$ *intersects* flow $(2,3)$.

Finally, as far as rate provisioning is concerned, we assume that a node's rate is no less than the sum of the sustainable rates of the flows traversing it, i.e. for every node $1\le h\le N$, $\sum_{(i,j):\, i\le h\le j} \rho_{(i,j)} \le R^h$. Note that this allows a node's rate to be utilized up to 100%, and it is therefore a necessary condition for stability. Moreover, we assume that the buffer of a node is large enough as to guarantee that traffic is never dropped.

4 The LUDB Methodology

In this paragraph, we briefly describe the *Least Upper Delay Bound* (LUDB) methodology [1,11]. We first explain it on nested tandems, and extend it to non-nested tandems later on. At a first level of approximation, LUDB consists in computing *all* the service curves for the tagged flow: we start from the aggregate service curves at each node, we apply Corollary 2 iteratively in order to remove one flow $(i,j) \not\equiv (1,N)$ from the tandem, and we convolve the service curves of nodes traversed by the same set of flows. Every time Corollary 2 is used, a new free parameter $s_{(i,j)}$ is introduced. Therefore, we compute in fact a *multi-dimensional infinity* of service curves. From each of these we can compute a delay bound for the tagged flow, hence the minimum among all the delay bounds is the *least upper delay bound.*

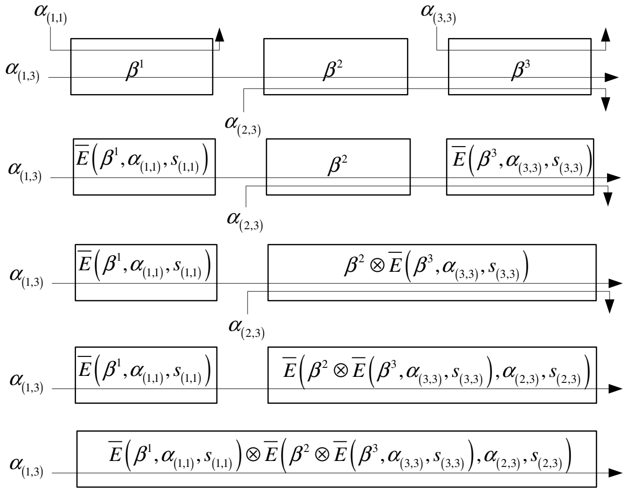

$\alpha_{(1,1)}$
$\alpha_{(3,3)}$
$\alpha_{(1,3)}$
β^1
β^2
β^3
$\alpha_{(2,3)}$
$\overline{E}(\beta^1,\alpha_{(1,1)},s_{(1,1)})$
$\overline{E}(\beta^3,\alpha_{(3,3)},s_{(3,3)})$
$\beta^2 \otimes \overline{E}(\beta^3,\alpha_{(3,3)},s_{(3,3)})$
$\overline{E}(\beta^2 \otimes \overline{E}(\beta^3,\alpha_{(3,3)},s_{(3,3)}),\alpha_{(2,3)},s_{(2,3)})$
$\overline{E}(\beta^1,\alpha_{(1,1)},s_{(1,1)}) \otimes \overline{E}(\beta^2 \otimes \overline{E}(\beta^3,\alpha_{(3,3)},s_{(3,3)}),\alpha_{(2,3)},s_{(2,3)})$

Fig. 2. An example of application of the LUDB methodology

For instance, Fig. 2 shows how to compute the set of end-to-end service curves for the tagged flow (1,3) in the nested tandem shown in Fig. 1a. The set of resulting (pseudoaffine) service curves $\pi^{\{1,3\}}$ depend on three parameters, $s_{(1,1)}$, $s_{(2,3)}$, $s_{(3,3)}$. More generally, let us consider a nested tandem of N nodes, whose level of nesting is $n \geq 2$ (otherwise the problem is trivial). The algorithm for computing the delay bound for the tagged flow is the following. As a first step, we build the *nesting tree* of the tandem, which is in fact a simplified representation of the tandem. Let us define two sets:

$$S_{(h,k)} = \{(i,j) : h \leq i \leq j \leq k \text{ and } l(i,j) = l(h,k)+1\},$$

i.e. the set of flows which are nested right into (h,k), and:

$$C_{(h,k)} = \{l : h \leq l \leq k \text{ and } \forall (i,j) \in S_{(h,k)}, l < i \text{ or } l > j\},$$

i.e. the set of nodes in path (h,k) that are not in the path of any flow in $S_{(h,k)}$. Note that, if $S_{(h,k)}=\varnothing$, then $C_{(h,k)}=\{h,h+1,...,k\}$.

For the sake of clarity, hereafter the nodes in the nesting tree are called *t-nodes*, in order to distinguish them from the nodes in the path of the tagged flow. In the nesting tree, there are two kind of t-nodes: non-leaf t-nodes represent *flows*, and leaf t-nodes represent *sets of nodes* in the path. More specifically, each non-leaf t-node contains a flow (h,k). The root t-node contains $(1,N)$. Furthermore, each t-node whose content is (h,k) has all flows $(i,j)\in S_{(h,k)}$ as direct descendants. Furthermore, if $C_{(h,k)}\neq\varnothing$, (h,k) has *one* more direct descendant representing $C_{(h,k)}$ (which is a leaf t-node). The level of nesting of a flow is the depth of the corresponding t-node in the nesting tree. Accordingly, we henceforth write that $(i,j)\rightarrow(h,k)$ iff $(i,j)\in S_{(h,k)}$, $S_{(h,k)}$ being the set of non-leaf *direct descendants* of (h,k), and that $(i,j)\rightarrow^{*}(h,k)$ to denote that (i,j) is a (possibly non-direct) descendant of (h,k). Fig. 3 shows the nesting tree of the tandem of Fig. 1a. Leaf t-nodes are shown as circles, while non-leaf nodes are ellipses. For instance, it is $(2,3)\rightarrow(1,3)$ and $(3,3)\rightarrow^{*}(1,3)$, whereas $(1,1)\not\rightarrow^{*}(2,3)$.

Once the nesting tree has been constructed, the set of end-to-end service curves for $(1,N)$ is computed by visiting the nesting tree from the leaves to the root as follows:

For each *leaf* t-node representing $C_{(h,k)}$ for some parent t-node (h,k), compute $\pi^{C_{(h,k)}}=\otimes_{j\in C_{(h,k)}}\beta^{j}$. Then, at a non-leaf t-node (h,k), compute a service curve as:

$$\pi^{\{h,k\}}=\pi^{C_{(h,k)}}\otimes\left[\bigotimes_{(i,j)\in S_{(h,k)}}\bar{E}\left(\pi^{\{i,j\}},\alpha_{(i,j)},s_{(i,j)}\right)\right], \tag{6}$$

i.e. as the convolution of i) the service curves obtained by applying Corollary 2 to the service curve computed at all child t-nodes, and ii) service curve $\pi^{C_{(h,k)}}$, if $C_{(h,k)}\neq\varnothing$ (otherwise assume for completeness that $\pi^{C_{(h,k)}}=\delta_0=\beta_{0,+\infty}$).

The set of end-to-end service curves for $(1,N)$, call it $\pi^{\{1,N\}}$, is obtained by computing the service curve at the root t-node. The LUDB for the tagged flow is the following:

$$V=\min_{\substack{s_{(i,j)}\geq 0,\\ (i,j)\rightarrow^{*}(1,N)}}\left\{h\left(\alpha_{(1,N)},\pi^{\{1,N\}}\left(s_{(i,j)}:(i,j)\rightarrow^{*}(1,N)\right)\right)\right\}. \tag{7}$$

Since $\pi^{\{1,N\}}$ is pseudoaffine and $\alpha_{(1,N)}$ is an affine curve, problem (7) is an optimization problem with a *piecewise linear* objective function. Computing the LUDB means solving a *piecewise-linear programming* (P-LP) problem.

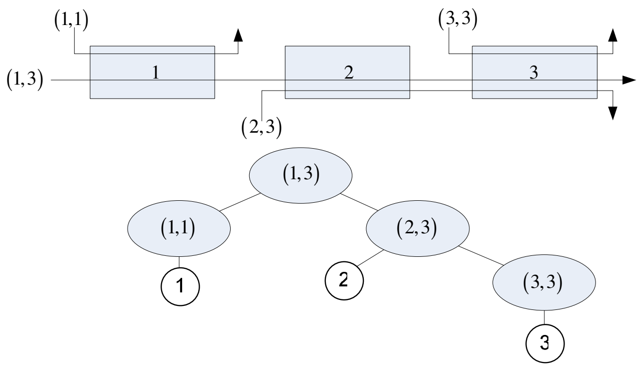


Fig. 3. A nested tandem and the related nesting tree

The LUDB methodology cannot be applied *directly* to non-nested tandems, such as the one shown in Fig. 1c. In fact, in that case, two flows intersect each other. In [11], it was observed that a non-nested tandem can always be *cut* into at most $\lceil N/2 \rceil$ *disjoint nested sub-tandems*. Therefore, one can use LUDB to compute partial, per sub-tandem delay bounds, and an end-to-end delay bound can be then computed by summing up the partial delay bounds. For instance, the tandem of Fig. 1c can be cut in two different ways, i.e. placing the cut *before* or *after* node 2. This way, two different end-to-end delay bounds can be computed, call them V^a and V^b, both using LUDB. Now, $V = V^a \wedge V^b$ is an end-to-end delay bound for the tagged flow, and both V^a and V^b can actually be the minimum, depending on the actual values of the nodes and flows parameters. As shown in [11], for each sub-tandem other than the first one, computing the *output arrival curve* for all the flows that traverse the cut is also required. For instance, if the cut is placed before node 2, the output arrival curve of flow (1,3) and (1,2) at the exit of node 1 are required in order to be able to analyze sub-tandem {2,3}. We proved in [11] that computing these output arrival curves is in fact identical to solving a simplified LUDB problem, and proposed algorithms to keep flows bundled as much as possible, for the sake of tightness. Therefore, the problem of computing bounds in non-nested tandem can be exploded into a number of LUDB computations, once an algorithm for computing *sets of cuts* is given.

5 DEBORAH

Hereafter, we overview the algorithms coded into DEBORAH [14] for computing upper bounds - in both nested and non-nested tandems - and those for computing lower bounds, which are common to both. We conclude the section by describing how to use the tool.

5.1 Nested Tandems

As shown in [11], P-LP problems such as (7) can be decomposed into a number of simplexes. Assume for ease of notation that:

$$\pi^{\{1,N\}} = \delta_D \otimes \left[\bigwedge_{1 \le x \le n} \gamma_{\sigma_x, \rho_x} \right],$$

where D and σ_x, $1 \le x \le n$, are linear functions of $s_{(i,j)}$, $(i,j) \to^* (1,N)$. Then, problem (7) can be formulated as follows:

$$\begin{array}{l} V = \min \left\{ D + \bigvee_{1 \le x \le n} \left[\dfrac{\sigma_{(1,N)} - \sigma_x}{\rho_x} \right]^+ \right\} \\ s.t. \\ s_{(i,j)} \ge 0, \qquad \forall (i,j) \to^* (1,N) \end{array} .$$

The above problem has a piecewise linear objective function, due to the maximum operator. It can however be decomposed into $n+1$ problems, as many as the terms in the max operator between square brackets (i.e., all the n leaky bucket stages of $\pi^{\{1,N\}}$, plus the null term given by $[\ ]^+$). In each sub-problem, the max is assumed to be achieved either for generic term x, $1 \le x \le n$, or for the null term, and the inequalities which are required for these assumptions to hold are added accordingly. We henceforth call each of those instances a *decomposition* of the original (P-LP) LUDB problem. The generic decomposition x, $1 \le x \le n$, and the $n+1$th one are shown below:

$$\begin{array}{ll} V_x = \min \left\{ D + \dfrac{\sigma_{(1,N)} - \sigma_x}{\rho_x} \right\} & \\ s.t. & \\ s_{(i,j)} \ge 0, & \forall (i,j) \to^* (1,N) \\ \dfrac{\sigma_{(1,N)} - \sigma_x}{\rho_x} \ge \dfrac{\sigma_{(1,N)} - \sigma_y}{\rho_y} & \forall y, 1 \le y \le n \\ \dfrac{\sigma_{(1,N)} - \sigma_x}{\rho_x} \ge 0 & \end{array} \quad \text{and} \quad \begin{array}{ll} V_{n+1} = \min \{D\} & \\ s.t. & \\ s_{(i,j)} \ge 0, & \forall (i,j) \to^* (1,N) \\ \dfrac{\sigma_{(1,N)} - \sigma_y}{\rho_y} \le 0 & \forall y, 1 \le y \le n \end{array} .$$

Then, the LUDB is computed as $V = \min_{1 \le x \le n+1} \{V_x\}$. The offset D and the bursts σ_x of the n stages of $\pi^{\{1,N\}}$ can either be *affine* functions of $s_{(i,j)} \ge 0$, $(i,j) \to^* (1,N)$, in which case all the decompositions are simplexes, or can themselves be piecewise linear functions of $s_{(i,j)} \ge 0$, $(i,j) \to^* (1,N)$. In this last case, however, they are obtained by composing sum and maximum operations recursively according to the nesting tree structure. Therefore, each problem can be recursively decomposed into a number of other problems, working out maxima and adding constraints at each recursive step, until the resulting problems turn out to be simplexes themselves. We call each such simplex a *recursive simplex decomposition* (RSD) of that LUDB problem. The problem with this approach is that the number of RSDs grows exponentially with the number of flows and nodes. As an example, for a case-study tandem with 30 nodes and 31 flows, nested up to level five, we obtain $\Omega = 1.5 \cdot 10^9$ RSDs.

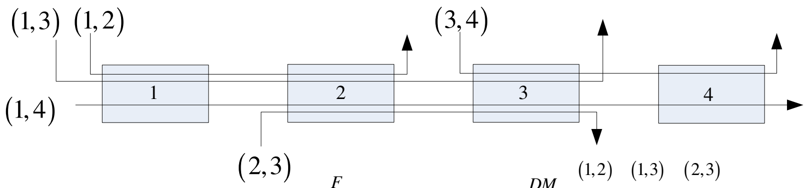
A much more efficient solution algorithm can be obtained by observing that most RSDs are *infeasible* and can be identified as such at a small cost. In fact, thanks to the recursive structure of the LUDB problem, it is fairly easy to identify small sets of infeasible constraints, each one of which may appear in possibly many RSDs. Once a set of constraints is identified as infeasible, all the RSDs which include that set can be safely skipped, reducing the overall number of simplexes to be solved to a much smaller figure. Thus, the key to a faster solution algorithm is to work *bottom-up* in the nesting tree: starting from the t-nodes immediately above the leaves, we compute the LUDB for the *sub-tree* rooted at each of them. In doing so, we check the feasibility of each resulting RSD, and we mark infeasible constraints or sets thereof. Moving upwards towards the root, at each father t-node we solve the LUDB problem, this time considering *only* those RSDs which do not include infeasible (sets of) constraints of child t-nodes. As soon as new infeasible RSDs are identified, they are marked and ruled out from then on. The bottom-up approach is considerably faster than a brute-force recursive simplex decomposition. For instance, in sink-tree tandems it reduces the number of simplexes from $M!$ to $O(M^2)$. In the previously mentioned case study, the overall number of RSDs to be solved or proved infeasible is reduced from $1.5 \cdot 10^9$ to $1.67 \cdot 10^6$. In this last case, DEBORAH finds the solution in less than 20 minutes on a 2.4GHz Intel Core2 E6600 processor.

5.2 Non-nested Tandems

With non-nested tandems, before computing the LUDBs for each sub-tandem, we have to *cut* it into a number of *nested* sub-tandems. Flows (i, j) can be stored in a *flow matrix* F, such that $F_{i,j} = 1$ is flow (i, j) exists. Interdependencies can be efficiently located by exploring F using simple bitwise operations. For instance, for flow (i, j) the interdependent flows are the 1s in $[1, i-1] \times [i, j-1] \cup [i+1, j] \times [j+1, N]$.

Let d be the number of such dependencies. As a first step, an $N \times d$ binary *Dependency Matrix* (*DM*) is computed. Fig. 4 shows a non nested tandem and its *F* and *DM*. For each couple of interdependent flows (i, j) and (h, k), the dependency is severed if we cut the tandem at any node in $[h, j+1]$. Accordingly, the DM has a 1 at all the rows $[h, j+1]$ for that dependency. Row n of the DM is thus the set of dependencies D_n which would be severed by cutting the tandem at node n. We are interested in finding sets of cuts such that: i) the resulting sub-tandems are all nested, ii) all the dependencies are satisfied, and iii) no cut can be eliminated without violating i). We call the above *Primary Sets of Cuts* (PSCs).

PSCs are those subsets of rows (i.e., nodes) such that at least a 1 can be found for every column (i.e. dependency), and no row dominates the other, i.e. $\{3\}$ and $\{2,4\}$ in the example of Fig. 4. The number of PSCs grows exponentially with the number of nodes, however with a small exponent: for a *dense* tandem, i.e. one including *all* possible flows, our experiments show that this number grows as $0.73 \cdot 2^{0.4N}$, i.e. about $3 \cdot 10^3$



F	1	2	3	4
1		×	×	×
2			×	
3				×
4				

DM	(1,2) (2,3)	(1,3) (3,4)	(2,3) (3,4)
1			
2	×		
3	×	×	×
4		×	×

Fig. 4. A non-nested tandem, its flow matrix (left) and dependency matrix (right)

for $N = 30$. There is unfortunately no way to tell which PSC will yield the smallest bound beforehand: intuitive principles like "minimum number of cuts/flows/overall sum of ρ s" may not yield the optimal result. Hence, all PSCs should be tried in principle. While this requires to compute exponentially many LUDBs, each LUDB is also exponentially simpler because sub-tandems are generally shorter due to the cutting, hence the computation time does not differ too much with respect to the nested tandem case. Furthermore, a considerable speedup (e.g., up to 80% for dense tandems of 30 nodes) can be harvested by *caching* partial per sub-tandem computations. It turns out, in fact, that the same sub-tandem can be common to many PSCs, hence computing its partial delay bound *once* (instead of once per PSC it belongs to) improves the efficiency. The exact procedure for computing LUDBs once PSCs are known is described in [11], to which the interested reader is referred to for the details.

5.3 Lower Bounds

As proved in [1], the LUDB is not always equal to the WCD. While the two are always equal in sink-tree tandems [10], this cannot be generalized to nested tandem, nor to fully nested tandems either. Therefore, it becomes important to compute also *lower bounds* on the WCD, so as to bound the region where the WCD is included, thus implicitly assessing how overrated the LUDB can be with respect to the WCD itself. As every *achievable* delay is itself a lower bound on the WCD, we heuristically setup a scenario where delays are large, so as to create a reasonably pejorative case. The algorithm for computing the lower bound is the following:

a) assume that all nodes are lazy, i.e. that $D^{(i)} = A^{(i)} \otimes \beta^{(i)}$ for each node *i*.

b) The tagged flow $(1, N)$ sends its whole burst $\sigma_{(1,N)}$ at time $t = 0$ and then stops. Therefore, the $\sigma_{(1,N)}$ th bit of the tagged flow experiences a larger delay than the other $\sigma_{(1,N)} - 1$, and that delay is our lower bound on the WCD.

c) Every cross flow (i, j) sends "as much traffic as possible" while the bits of the tagged flow are traversing the node, so as to delay the $\sigma_{(1,N)}$ th bit of the tagged flow. Among the infinite Cumulative Arrival Functions (CAFs) that follow this rule, we pick the *greedy* one, i.e. one such $A_{(i,j)}(t-t_0)=\alpha_{(i,j)}(t-t_0)$, t_0 being the time instant where the *first* bit of the tagged flow arrives at node *i*, and the *delayed-greedy* one, where cross flow (i,j) sends the whole burst *just before* the last bit of the tagged flow arrives. It turns out that, depending on the values associated to the nodes and flows parameters, using either greedy or delayed greedy CAFs for the cross flows actually leads to different delays, and it is not always possible to establish which is the largest beforehand.

Within DEBORAH, each CAF is represented as a piecewise linear function, not necessarily convex. This allows us to *approximate* any kind of curve. Furthermore, the curves that we use are themselves piecewise linear, so their representation is exact. Each CAF $A^k_{(i,j)}$ is a list of $Q^k_{(i,j)}$ breakpoints; each breakpoint B_i is represented through its Cartesian coordinates t_i, b *and* a *gap* g_i, i.e. a vertical discontinuity which allows for instantaneous bursts: $B_i=(t_i,b_i,g_i)$, $A^k_{(i,j)}\equiv\left\{B_x,\ 1\le x\le Q^k_{(i,j)}\right\}$. As the CAFs are wide-sense increasing, the abscissas of the breakpoints are strictly increasing, and $b_{i+1}\ge b_i+g_i$. The number of breakpoints is finite. This is because, since the worst-case delay stays finite, then it is achieved for sure in finite time, and therefore we can safely assume that the CAF remains constant after the last breakpoint. For instance, an affine CAF with initial burst σ and a constant slope ρ up to time τ is represented as $\{(0,0,\sigma),(\tau,\rho\cdot\tau+\sigma,0)\}$.

The operations required for computing the CDF of the tagged flow at node N are:

1. *FIFO multiplexing* of several CAFs at the entrance of a node, so as to compute the aggregate CAF.
2. *Convolution* between the aggregate CAF and a node's rate-latency service curve, i.e. computation of a lower bound for the aggregate CDF.
3. *FIFO de-multiplexing* of flows at the exit of a node, i.e. computation of *per-flow* CDFs from the aggregate CDF. This is required to take into account flows leaving the tandem.

The multiplexing is a summation of CAFs, which boils down to computing the union of the respective breakpoints and summing their ordinates. The convolution algorithm is explained in [3], Chapter 1.3, in its most general form. Our implementation capitalizes on the service curve being latency-rate and on the CAF being piecewise linear. In this case, all it takes is comparing the slope of the linear pieces in the CAF against the rate of the service curves, and computing intersections. The resulting CDF has a different set of breakpoints with respect to the CAF, and it is *continuous*, even if the CAF is not, since the service curve is itself continuous. The third operation, i.e. FIFO de-multiplexing, exploits the FIFO hypothesis: more specifically, for all flows (i,j) traversing node k as part of a FIFO aggregate, $\forall t\ge 0$ $D^k_{(i,j)}(t)=A^k_{(i,j)}(x(t))$, where $x(t)=\sup\{\tau\le t: A^k(\tau)=D^k(t)\}$. Let R^{out} be the rate of $D^k(t)$ in $[t_1,t_2)$. If $A^k(t)$ is continuous in $[x(t_1),x(t_2))$, call R^{in} and $R^{in}_{(i,j)}$ its rate and that of $A^k_{(i,j)}(t)$ in that interval. Then, the rate of $D^k_{(i,j)}(t)$ in $[t_1, t_2)$ is equal to $R^{out}_{(i,j)} = R^{in}_{(i,j)} \cdot R^{out}/R^{in}$. If instead $A^k(t)$ has a discontinuity in t due to flow f 's burst, so that $A^k(t^-) = b_0$ and $A^k(t^+) = b_1 > b_0$, then all the traffic in the aggregate CDF in $\left[\left(D^k\right)^{-1}(b_0), \left(D^k\right)^{-1}(b_1)\right)$ belongs to flow f. Thus, if R^{out} is the rate of $D^k(t)$ in that interval, the rate of $D^k_{(i,j)}(t)$ in the same interval is equal to $R^{out} \cdot 1_{\{(i,j)\equiv f\}}$. Fig. 5 reports a graphic representation of the FIFO multiplexing and de-multiplexing of two CAFs.

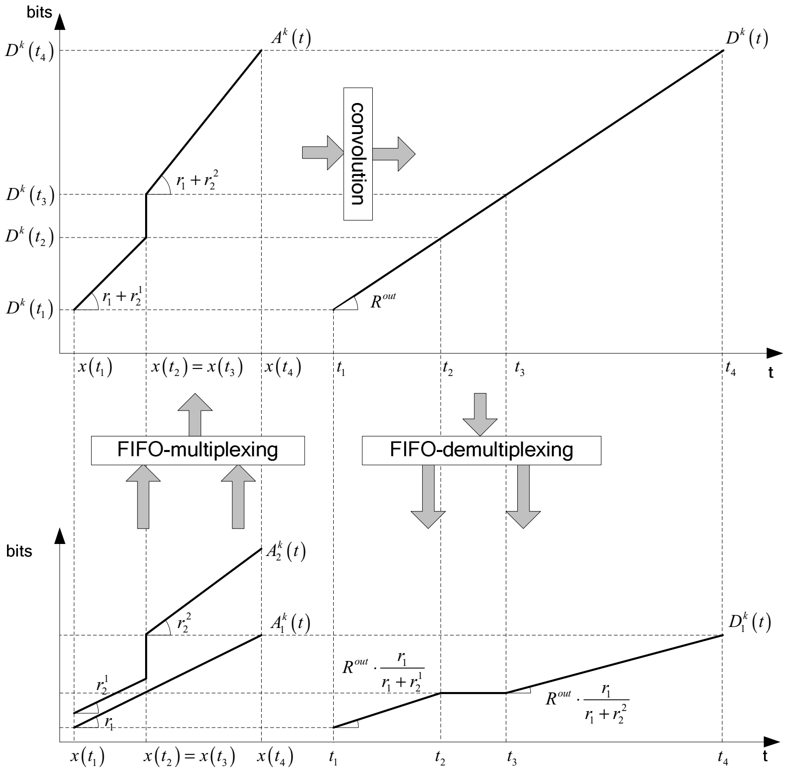


Fig. 5. Basic operations at a FIFO node

Other tools that have been developed for Network Calculus. The DISCO calculator [12] computes *output arrival curves* from (concave) input arrival curves, assuming *blind* multiplexing (instead of FIFO). The COINC library [13] implements basic (min, +) algebra operations, hence – although not a tool itself, lacking network representation, it can be used to build a tool. The RTC [18] and CyNC [19] toolboxes allow one to compute CDFs from CAFs through (min,+) convolution. However, as far as we know, they cannot compute LUDBs in FIFO systems, and we are not aware that they implement demultiplexing of flows at the output of a FIFO server, which is necessary for our lower bound analysis.

5.4 Using DEBORAH

In this section we show how to use DEBORAH for analyzing user-defined network topologies. DEBORAH is a command-line program written in portable C++, which can be compiled for a number of architectures; so far it has been successfully run on Linux, Windows and MacOS X. Its arguments can be classified into three functional categories: a) specification of the tandem topology; b) indication of the desired computation (currently LUDB, lower bound and per-node upper bound); c) network provisioning modifiers, e.g. to scale the rates assigned to flows or nodes by a constant factor or to explicitly select the tagged flow. We illustrate the most significant features offered by the tool via a couple of practical examples. As a first case we consider a nested tandem of 10 nodes and 11 flows as shown in Fig, 6. The service curves at each node and the arrival curves of each flow are specified in Table 1 and 2.

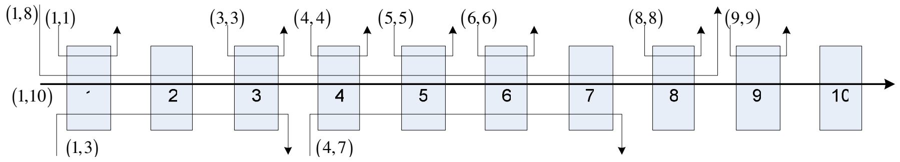


Fig. 6. Case-study nested tandem

The tandem topology is input in a text file (say "`ex1.conf`") using a straightforward syntax. The file must begin with the directive `TANDEM` N F, which denotes a tandem with N nodes and F flows. Next, the service curve of each node is configured by means of a "`NODE` n θ R" line, where n is the node ID from 1 to N, and θ, R are its latency and the rate respectively. Similarly, flows are specified using "`FLOW` i j σ ρ", where i, j are the source and sink nodes and σ, ρ are the flow's leaky bucket parameters. The tagged flow is automatically selected as the one spanning the longest segment (usually the whole tandem), or it can be manually specified by using `TFLOW` instead of `FLOW` in its declaration. Apart from the `TANDEM` directive, which is expected to come first in the file, the other lines can appear in just any order. Lines beginning with a hash (#) character are treated as comments and ignored.

A tandem configuration file is normally the first command line argument. If no other arguments are specified, DEBORAH parses the topology, performs some sanity checks (e.g. checks that the nodes' rates are sufficient) and prints a report. For nested tandems, for instance, it will print the associated nesting tree using a text notation.

The LUDB and the lower bound are computed by specifying the `-ludb` and `-lb` options after the configuration file name: `./deborah ex1.conf -ludb [-lb]`.

Regarding the LUDB, the tool reports detailed information including the numeric value of the optimal $s_{(i,j)}$ parameters and the symbolic expression of the service curve, and performance figures such as the number of simplexes evaluated and the total computation time. By default, DEBORAH runs the exact LUDB algorithm described in this paper. A heuristic approximation (not described here for reasons of space, see [17]) can be requested using the `-ludb-heuristic k` option, where k is the maximum number of randomly-selected RSDs used at each node in the nesting tree.

Table 1. Node provisioning

Node ID	θ, R
1	0.3, 70
2	0.2, 10
3	0.1, 40
4	0.1, 40
5	0.2, 60
6	0.1, 55
7	0.2, 7
8	0.3, 70
9	0.1, 60
10	0.2, 10

Table 2. Flow parameters

(i,j)	σ, ρ
1, 10	200, 1
1, 1	100, 60
1, 3	200, 3
3, 3	2000, 30
1, 8	100, 2
4, 4	400, 30
4, 7	300, 2
5, 5	300, 50
6, 6	200, 45
8, 8	100, 55
9, 9	300, 55

When LUDB computation is invoked, the tool detects the tandem to be non-nested and sets to cutting the tandem into multiple disjoint sub-tandems. Each computed PSC is reported in the program output along with the associated delay bound, the minimum of which is elected as the LUDB. As the critical performance factor here is represented by the possibly large number of PSCs, the latter can be controlled with the `-ludb-cuts-len L` option, which throws away PSCs exceeding the shortest one by more than *L* cuts. In fact, as *L* grows larger, a diminishing likelihood of finding good bounds can be observed. Finally, per-node bounds can be computed, using `-per-node`. The latter can be used as a baseline, as they are generally largely overrated.

For the lower bounds, DEBORAH prints the number of flow combinations analyzed versus the maximum possible, as well as the computation time. Again, the tool provides an option to deal with performance scalability issues by adding `–lb-random-combo p` to the command line, which forces the total number of combinations to be computed to stay below *p%* of the theoretical limit.

While it is easy to create and analyze custom topologies using text files, DEBORAH provides means to generate particular classes of tandems in an analytical way, which can be useful to conduct systematic studies. Specifically, it is possible to generate nested topologies whose nesting tree is a balanced trees of any order and depth, and non-nested tandems populated with an arbitrary number of flows. For the first case, the syntax is: `./deborah –gen-tree O K file.conf` where *O* is the order of the tree (the number of children for each node), *K* is the tree depth and *file.conf* is the name of the file where the configuration will be stored. Nodes and flows are provisioned according to stochastic variables which can be controlled using dedicated switches. For non-nested tandems, command `-gen-nnested N F file.conf` is used, where *N* is the number of nodes and *F* is the *percentage* of flows (randomly selected) with respect to a maximum of $N \cdot (N-1)/2$.

Finally, loaded configurations can be altered before processing takes place. For instance it is possible to override the tagged flow ID with `-tagged N`, or to scale the rates by a given factor simultaneously with `-scale-rates Rf Rn`, where the rates of flows and nodes are multiplied by *Rf* and *Rn* respectively.

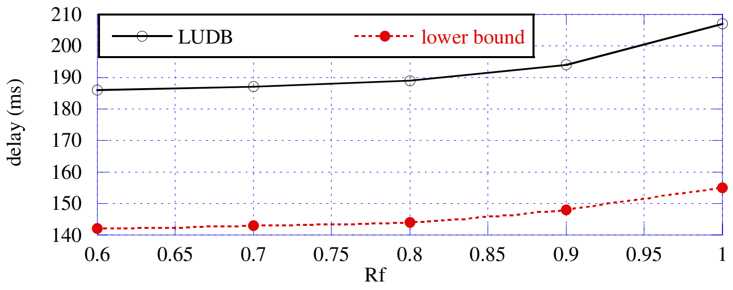


Fig. 7. LUDB / lower-bound comparison

As an example, Fig. 7 reports the LUDB and lower bound as a function of the flow rates for the nested tandem of Fig. 6. The figure was obtained using, a simple BASH script to repeatedly invoke DEBORAH with the different values for *Rf*. Computation times (subject to the coarse precision of Linux timers) are feebly, if at all, affected by the load, and they are 10ms and 40ms for the upper and lower bounds respectively.

6 Conclusions

This paper presented DEBORAH, a tool for computing upper and lower bounds on the worst-case delay in FIFO tandems using Network Calculus. We described the algorithms used for i) computing the LUDB in nested tandems, ii) cutting a non-nested tandem into nested sub-tandems, so as to enable partial LUDB computations, and iii) computing lower bounds on the worst-case delay. We have also described how to use the tool for analysis. To the best of our knowledge, DEBORAH is the first tool to compute bounds on the delay in FIFO tandem networks.
Future work on the tool will consider employing integer fractions – instead of floating points – to model slopes in the computation of the lower bounds, so as to reduce the impact of floating point arithmetic errors in the computations.

References

1. Bisti, L., Lenzini, L., Mingozzi, E., Stea, G.: Estimating the Worst-case Delay in FIFO Tandems Using Network Calculus. In: VALUETOOLS 2008, Athens, GR, October 21-23 (2008)
2. Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W.: An Architecture for Differentiated Services. IETF RFC 2475 (1998)
3. Le Boudec, J.-Y., Thiran, P.: Network Calculus. LNCS, vol. 2050. Springer, Heidelberg (2001)
4. Rosen, E., Viswanathan, A., Callon, R.: Multiprotocol Label Switching Architecture. IETF RFC 3031 (January 2001)
5. Cruz, R.L.: A calculus for network delay, part i: Network elements in isolation. IEEE Transactions on Information Theory 37(1), 114–131 (1991)
6. Cruz, R.L.: A calculus for network delay, part ii: Network analysis. IEEE Transactions on Information Theory 37(1), 132–141 (1991)

7. Agrawal, R., Cruz, R.L., Okino, C., Rajan, R.: Performance Bounds for Flow Control Protocols. IEEE/ACM Trans. on Networking 7(3), 310–323 (1999)
8. Chang, C.S.: Performance Guarantees in Communication Networks. Springer, New York (2000)
9. Lenzini, L., Mingozzi, E., Stea, G.: Delay Bounds for FIFO Aggregates: a Case Study. Elsevier Computer Communications 28(3), 287–299 (2005)
10. Lenzini, L., Martorini, L., Mingozzi, E., Stea, G.: Tight End-to-end Per-flow Delay Bounds in FIFO Multiplexing Sink-tree Networks. Performance Evaluation 63, 956–987 (2006)
11. Lenzini, L., Mingozzi, E., Stea, G.: A Methodology for Computing End-to-end Delay Bounds in FIFO-multiplexing Tandems. Performance Evaluation 65, 922–943 (2008)
12. Schmitt, J.B., Zdarsky, F.A.: The DISCO Network Calculator - A Toolbox for Worst Case Analysis. In: Proc. of VALUETOOLS 2006, Pisa, Italy. ACM, New York (October 2006)
13. Bouillard, A., Thierry, É.: An Algorithmic Toolbox for Network Calculus. Journal of Discrete Event Dynamic Systems 18(1), 3/49 (2008)
14. Website of the Computer Networking Group at the University of Pisa, continuously updated, `http://cng1.iet.unipi.it/wiki/index.php/Deborah`
15. Bouillard, A., Johuet, L., Thierry, E.: Tight performance bounds in the worst-case analysis of feed-forward networks. In: Proc. INFOCOM 2010, San Diego, US, March 14-19 (2010)
16. Fidler, M.: A Survey of Deterministic and Stochastic Service Curve Models in the Network Calculus. IEEE Communications Surveys and Tutorials 12(1), 59–86 (2010)
17. Bisti, L., Lenzini, L., Mingozzi, E., Stea, G.: Computation and Tightness assessment of delay bounds in FIFO-multiplexing tandems. Technical Report, Pisa (May 2010), `http://info.iet.unipi.it/~stea/`
18. Wandeler, E., Thiele, L.: Real-Time Calculus (RTC) Toolbox,(2006), `http://www.mpa.ethz.ch/Rtctoolbox`
19. Schioler, H., Schwefel, H.P., Hansen, M.B.: CyNC – a MATLAB/Simulink Toolbox for Network Calculus. In: Proc. VALUETOOLS 2007, Nantes, FR (October 2007)