Computation and Tightness Assessment of End-to-end Delay Bounds in FIFO-multiplexing Tandems[1]

Luca Bisti, Luciano Lenzini, Enzo Mingozzi, Giovanni Stea

Dipartimento di Ingegneria dell'Informazione, University of Pisa, Italy
Via Diotisalvi, 2 56122 Pisa, Italy - Ph. +39 050 2217599

{l.bisti, l.lenzini, e.mingozzi, g.stea}@iet.unipi.it

Technical Report, May 2010

ABSTRACT

This paper addresses the problem of estimating the worst-case end-to-end delay for a flow in a tandem of FIFO multiplexing nodes, following up our previous work [14]. We consider the state-of-the-art method for computing *delay bounds*, i.e. upper bounds on the worst-case delay, called the *Least Upper Delay Bound* (LUDB) methodology, and we describe efficient numerical techniques to compute the LUDB. The latter allow good delay bounds to be computed for tandems of several tens of nodes within minutes of computation time. Furthermore, we show that, unlike what happens in some specific sub-classes of FIFO tandems analyzed in the previous work, the LUDB may actually be larger than the worst-case delay, even when end-to-end analysis is possible. Therefore, in order to assess how close the derived bounds are to the actual, still unknown, worst-case delays, we devise a method to compute *lower bounds* on the worst-case delay. Our analysis shows that the gap between the upper and lower bounds is reasonably small, at least when end-to-end analysis is possible, which implicitly validates the upper bounds themselves.

Keywords

Network Calculus, FIFO-multiplexing, Delay Bound.

1. INTRODUCTION

Network calculus ([5], [7]-[10]) is a theory for deterministic network performance analysis. Originally devised to address Quality of Service problems in IP networks, it has also found fields of applications in several other areas, including wireless sensor networks [22]-[23], Ethernet installations [24], and Systems-on-Chip [25]. Its main feature is its ability to compute performance bounds, such as *delay bounds*, which are useful to assess the capabilities of a network architecture to support real-time applications. As far as IP networks are concerned, Network Calculus has been used in the previous decade to compute delay bounds for networks employing *per-flow* resource management, the most notable case being that of the IETF Integrated Services (IntServ) architecture (see, for example, [5], Chapter 2). More recently, network architectures employing *per-aggregate* resource management have become a reality, due to their better scalability. Two noticeable examples of architectures employing per-aggregate resource management are Differentiated Services (DiffServ [4]), and Multi-Protocol Label Switching (MPLS, [6]), both standardized by the IETF. In the former, flows traversing a domain are aggregated (or *multiplexed*) in a small number of classes or Behavior Aggregates (BA), whose forwarding treatment is standardized, and QoS is provisioned on a per-aggregate basis at each node. In the latter, flows are aggregated into Forwarding Equivalence Classes (FECs) and forwarding and routing are performed on a per-FEC basis.

Computing good delay bounds in networks employing per-aggregate resource management is however particularly challenging. Obviously enough, a delay bound is as good as it is *tight*, i.e. close to the actual maximum delay that can theoretically be experienced by a bit of the flow. We refer to the latter as the *worst-case delay* (WCD). While it is fairly simple to compute the WCD under per-flow resource management, computing it in networks employing per-aggregate resource management appears to be considerably more complex. During the last decade, several results have appeared in the literature on this subject. The aim of these works is to compute *delay bounds* in *feed-forward* networks, which are known to be stable for any utilization below 100% [5]. For instance, [18] presents tools and techniques for computing end-to-end delay bounds for flows in feed-forward networks of *blind* multiplexing nodes. "Blind" means that no assumption is made regarding the flow multiplexing criterion: for instance, both a FIFO multiplexing scheme and a strict priority multiplexing scheme in which the *tagged flow* (i.e., the one being analyzed) is always multiplexed at the lowest priority fit this definition. Smaller bounds can be obtained by explicitly assuming that a FIFO multiplexing scheme is in place at the node. As regards FIFO multiplexing, some recent works [11]-[14] 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. It entails: i) applying the above theorem iteratively, so as to obtain a *parametric set of end-to-end service curves* for a flow, ii) computing a parametric expression for the delay bound, and iii) minimizing over the set of parameters so as to obtain, in fact, the least upper bound. The term "least", of course, refers to the bounds that

[1] Part of the content of this report has appeared as [1] and [2].

can be found using this method. End-to-end analysis and global minimization are the two points of strength of LUDB. As shown in [14], we are actually able to derive end-to-end service curves only for a particular class of tandems, called *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.

There are currently two open issues related to the LUDB methodology. The first one is that, except for a very limited number of (nested) tandem topologies, no closed-form solution for the LUDB exists. More specifically, [11] presents a closed-form solution LUDB for a tagged flow traversing a tandem of nodes with one-hop persistent cross traffic, while [12] presents a formula for *sink-tree* tandems. While it is theoretically possible that the formulas computed in [11]-[12] can be generalized, in this paper we follow a numerical approach. As shown in [14], the computation of the LUDB is a *piecewise linear problem*, since it exhibits a piecewise linear objective function and a number of linear constraints. In this paper we show that the problem can be solved by decomposing it recursively into a number of simplexes. However, the number of simplexes to be solved may grow very fast, making a brute-force approach computationally infeasible when the number of flows approaches few tens. Therefore, we first identify properties that greatly reduce the amount of computations for these medium-sized problems (i.e., up to few tens of nodes), and then we present an effective heuristics to obtain good approximated solutions at a low computational cost for larger-scale problems (i.e., several tens of nodes). For non-nested tandems, we first identify a much smaller number of sets of cuts that are candidate to produce the best delay bounds, called the *Primary Sets of Cuts (PSCs)*. Since it is impossible to define *a priori* which PSC will produce the tightest delay, we need to compute the delay bound for all of them. However, we exploit the problem structure to obtain an efficient implementation, where the number of operations to be computed grows sublinearly with the number of PSCs. Our analysis shows that delay bound computation in non-nested tandems is generally more complex than in nested ones, but still computationally affordable for paths of 30 nodes or slightly more, i.e. as long as the longest paths in today's planetary Internet [26]. Furthermore, our bounds are much smaller than cumulative per-node delay bounds, the gap between them increasing exponentially with the number of nodes, which justifies the increased complexity.

The second open issue is *tightness*. LUDB is shown to yield *tighter* bounds with respect to both per-node analysis, and another end-to-end methodology, described in [15], which does not use global minimization. However, it is still unknown, in the general case, how far the bounds thus computed are from the WCD. For *sink-tree* tandems, which are nested ones, it was proved in [12] that the LUDB is actually *equal* to the WCD. However, whether this is true in more general settings is still an open question, made particularly challenging by the fact that no better computation method is available. In particular, since it is commonly believed that end-to-end analysis is a *necessary* condition for computing good delay bounds, one might wonder if it is also *sufficient*, i.e. whether the LUDB is tight in *all* nested tandems. In this paper, we provide a negative answer to the above question. The LUDB method may actually yield *loose* bounds even in very simple nested tandems. We prove this by counterexample: we devise a method, called *Flow Extension,* that can be used in conjunction to the LUDB methodology, so as to compute *smaller* delay bounds than those computed through LUDB alone, at least in some cases. This result is significant for two reasons: on one hand, it may sometimes lead to improved delay bounds; on the other hand, its significance from a theoretical standpoint lies in proving that the Network Calculus theorem that is at the core of the LUDB method is not always sufficient to describe the worst-case behavior of FIFO networks. This said, assessing *how tight* the computed bounds are becomes an important issue. Being unable so far to identify a provable worst-case scenario, we propose heuristics to approximate it. More specifically, we construct a set of scenarios where a flow experiences a *large* delay, which is itself a *lower bound* on the WCD, and we provide an algorithm to efficiently compute this lower bound. The interval between the lower and the upper bounds serves as an estimate of the tightness of the upper bound itself.

The algorithms described in this paper for computing upper and lower bounds on the WCD have been implemented in a software tool, called DEBORAH (DElay BOund Rating AlgoritHm, [20]), which is publicly available. This is the first software of this kind, to the best of our knowledge.

The rest of the paper is organized as follows: Section 2 reports some background on Network Calculus, also introducing some of the notation that will be used throughout the paper. In Section 3 we give a formal problem statement; we describe the LUDB methodology and efficient numerical methods to solve it in Section 4. In Section 5 we prove that the LUDB may actually be larger than the WCD, also describing how to compute smaller bounds than the LUDB. In Section 6, we present an algorithm for computing a lower bound on the WCD, using which we assess the tightness of the LUDB in some non-trivial case studies. Section 7 briefly describes how to use the DEBORAH tool. Finally, conclusions are reported in Section 8.

2. NETWORK CALCULUS BACKGROUND

This section introduces basic Network Calculus concepts, using the same notation as in [5]. Subsection 2.1 explains the framework for modeling FIFO rate-latency nodes traversed by leaky-bucket shaped flows that we developed in [12]-[14].

In Network Calculus, data flows are described by means of a wide-sense increasing cumulative function $R(t)$, defined as the number of bits seen on the flow in time interval $[0,t]$. Specifically, let $A(t)$ and $D(t)$ be the *Cumulative Arrival* and *Cumulative Departure* functions (CDA and CDF) characterizing the same data flow before and after a network element, respectively. Then, the network element can be modeled by the *service curve* $\beta(t)$ if

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

for any $t \geq 0$. The flow is said to be guaranteed the (minimum) service curve β. The infimum on the right side of (1), as a function of t, is called the min-plus convolution of A and β, and is denoted by $(A\otimes\beta)(t)$. Min-plus convolution is commutative and associative. Furthermore, convolution of concave curves is equal to their minimum. Several network elements, such as delay elements, links, and regulators, can be modeled by corresponding 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 packet schedulers can be modeled through *rate-latency* service curves, defined as follows:

$$\beta_{\theta,R}(t) = R \cdot [t-\theta]^+$$

for some $\theta \geq 0$ (the latency) and $R \geq 0$ (the rate). Notation $[x]^+$ denotes $\max\{0, x\}$. A fundamental result of Network Calculus is that the service curve of a tandem of network elements traversed by a data flow is obtained by convolving the service curves of each network element.

Guaranteeing performance bounds to traffic flows requires that the arrivals be constrained through *arrival curves*. A wide-sense increasing function α is an arrival curve for a flow characterized by a CAF A if it is:

$$A(t) - A(\tau) \leq \alpha(t-\tau) \text{, for all } \tau \leq t.$$

As an example, 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.

By combining together arrival and service curve characterizations of data traffic and network elements, respectively, it is possible to derive relevant performance bounds. In fact, assume that an element (or tandem of elements) is characterized by a service curve β and that a flow traversing that node is constrained by the arrival curve α. Then, if the node serves the bits of this flow in FIFO order, the delay is bounded by the horizontal deviation

$$h(\alpha,\beta) \triangleq \sup_{t\geq 0}\left[\inf\left\{d \geq 0 : \alpha(t-d) \leq \beta(t)\right\}\right] \qquad (2)$$

Intuitively, h is the amount of time the curve α must be shifted forward in time so that it lies below β. From (2) it follows that $\beta_1 \leq \beta_2 \Rightarrow h(\alpha,\beta_1) \geq h(\alpha,\beta_2)$. Notation $\beta_1 \leq \beta_2$ means that $\forall t \; \beta_1(t) \leq \beta_2(t)$.

A well-known result related to a tandem of N rate-latency nodes β_{θ^i,R^i}, $1 \leq i \leq N$, traversed by a $\gamma_{\sigma,\rho}$-constrained flow follows from (2), i.e., the end-to-end delay bound is given by

$$d = \sum_{i=1}^{N}\theta^i + \sigma \Big/ \bigwedge_{1\leq i\leq N}\left\{R^i\right\} \qquad (3)$$

provided that $\rho \leq R^i$ for any i. Notation $\wedge$ denotes the minimum operation. It is shown in [5] that bound (3) is actually achievable (i.e., it is the worst-case delay), at least if the arrival curve of the flow and the service curves of all the nodes are actually tight characterizations. Henceforth, we will always assume that this hypothesis is verified.

2.1 FIFO Multiplexing

Regarding FIFO multiplexing, a fundamental result, first derived in [9], is reported in [5], Chapter 6. Assume that two flows are FIFO multiplexed into the same network element, characterized by service curve β. Assume that α_2 is an arrival curve for flow 2. Then, the service received by flow 1 can be determined by computing its *equivalent* service curve $\beta_1^{eq}(t,\tau)$, as follows.

Theorem 2.1 (FIFO Minimum Service Curves [5]).
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:

$$\beta_1^{eq}(t,\tau) = \left[\beta(t) - \alpha_2(t-\tau)\right]^+ \cdot 1_{\{t>\tau\}}$$

For any $\tau \geq 0$ such that $\beta_1^{eq}(t,\tau)$ is wide-sense increasing, then flow 1 is guaranteed the (equivalent) service curve $\beta_1^{eq}(t,\tau)$.

Theorem 2.1 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. For ease of notation, we write $E(\beta,\alpha,\tau)(t)$ to denote the equivalent service curve obtained from applying Theorem 2.1 to a service curve $\beta(t)$, by subtracting from it arrival curve $\alpha(t-\tau)$. 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.

As an example, if the node is a rate-latency one, i.e. $\beta = \beta_{\theta,R}(t)$, and flow 2 is leaky-bucket shaped, i.e. $\alpha_2(t) = \gamma_{\rho_2,\sigma_2}(t)$, then Theorem 2.1 yields the following set of equivalent service curves for flow 1.

$$E(\beta,\alpha_2,\tau)(t) = (R-\rho_2)\left[t - \left(\theta + \frac{\sigma_2 + \rho_2(\theta-\tau)}{R-\rho_2}\right)\right]^+ 1_{\{t>\tau\}} \qquad (4)$$

The curves are also shown in Figure 1, from which the following two observations can be made:

a) $E(\beta,\alpha_2,\tau)$ is not necessarily a rate-latency curve. More specifically, it can be either a rate-latency curve (if $\tau \leq \theta + \sigma_2/R$) or a different kind of curve, namely an affine curve shifted to the right (if $\tau > \theta + \sigma_2/R$).

b) not all the curves obtained from Theorem 2.1 are actually relevant. For instance, all the curves obtained for $\tau < \theta + \sigma_2/R$ lie entirely below the one obtained for $\tau = \theta + \sigma_2/R$, and are therefore useless for computing performance bounds.

It has been proved in [12]-[14] (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:

$$\pi = \delta_D \otimes \left[\bigotimes_{1\leq x\leq n}\gamma_{\sigma_x,\rho_x}\right] \qquad (5)$$

i.e., as a multiple affine curve shifted to the right. Note that, since affine curves are concave, (5) is equivalent to:

$$\pi = \delta_D \otimes \left[\bigwedge_{1\leq x\leq n}\gamma_{\sigma_x,\rho_x}\right] \qquad (6)$$

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 $\beta_{\theta,R} = \delta_\theta \otimes \gamma_{0,R}$. A three-stage pseudoaffine curve is shown in Figure 2.

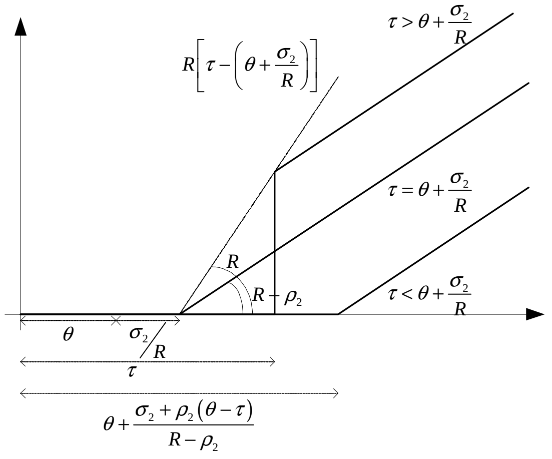
The alert reader will notice that, for any value of τ, all the curves obtained from (4) are pseudoaffine. Although more general than rate-latency curves, pseudoaffine curves are still fairly easy to manage from a computational standpoint. The following two properties, proved in [12], will be used throughout this paper:

Property 2.2 (closeness with respect to convolution):
The convolution of two pseudoaffine curves is a pseudoaffine curve, whose offset is the sum of the offsets of the operands, and whose leaky-bucket stages are the union of the leaky-bucket stages of both operands.

Property 2.3 (delay bound):
Let π be a pseudoaffine curve, with offset D and n leaky-bucket stages γ_{σ_x,ρ_x}, $1 \le x \le n$, and let $\alpha = \gamma_{\sigma,\rho}$. If $\rho_\pi^* \ge \rho$, then: $h(\alpha,\pi) = \pi^{-1}(\sigma)$, where $\pi^{-1}(\bullet)$ denotes the pseudo-inverse of $\pi(\bullet)$, defined in [5]. Moreover:

$$h(\alpha,\pi) = \pi^{-1}(\sigma) = D + \bigvee_{1\le x\le n}\left[\frac{\sigma-\sigma_x}{\rho_x}\right]^+, \tag{7}$$

where $\vee$ denotes the maximum operator.



$\tau > \theta + \frac{\sigma_2}{R}$
$R\left[\tau - \left(\theta + \frac{\sigma_2}{R}\right)\right]$
$\tau = \theta + \frac{\sigma_2}{R}$
R
$R+\rho_2$
$\tau < \theta + \frac{\sigma_2}{R}$
θ
σ_2/R
τ
$\theta + \frac{\sigma_2 + \rho_2(\theta-\tau)}{R-\rho_2}$

Figure 1 - The set of equivalent service curves for flow 1

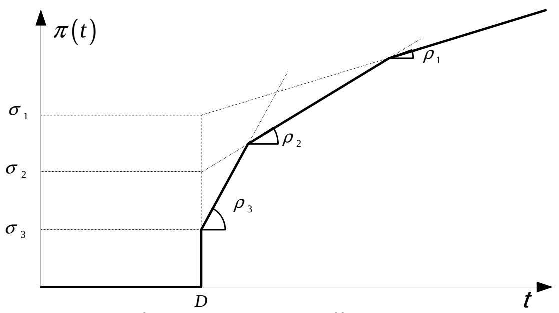


Figure 2 - Example of a three-stage pseudoaffine curve

Finally, Theorem 2.1 can be specialized for the case of pseudoaffine service curves and leaky-bucket arrival curves as follows:

***Corollary 2.4* ([12]):**
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, which are served in FIFO order, 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:

$$\overline{E}(\pi,\alpha,s) = \delta_{D+\bigvee_{1\le i\le n}\left[\frac{\sigma-\sigma_i}{\rho_i}\right]^+ + s} \otimes \left[\bigotimes_{1\le x\le n} \gamma_{\rho_x\left\{s+\bigvee_{1\le i\le n}\left[\frac{\sigma-\sigma_i}{\rho_i}\right]^+ - \frac{\sigma-\sigma_x}{\rho_x}\right\},\rho_x-\rho}\right], \tag{8}$$

or, equivalently,

$$\overline{E}(\pi,\alpha,s) = \delta_{h(\alpha,\pi)+s} \otimes \left[\bigotimes_{1\le x\le n} \gamma_{\rho_x\{s+h(\alpha,\pi)-D\}-(\sigma-\sigma_x),\rho_x-\rho}\right] \tag{9}$$

are pseudoaffine equivalent service curves for flow 1.

It can be proved that the set $S \triangleq \left\{\overline{E}(\pi,\alpha,s),\, s \ge 0\right\}$ is a proper *subset* of $\mathrm{T} = \left\{E(\beta,\alpha,\tau),\, \tau \ge 0\right\}$, i.e. it does not include some equivalent service curve that would be computed through Theorem 2.1. However, it does include those equivalent service curves which are relevant for computing delay bounds. More specifically, for each curve $x \in \mathrm{T} \setminus S$, there exists a curve $y \in S$ such that $y \ge x$. Therefore, all the "good" performance bounds that can be found by applying Theorem 2.1 can also be found by applying Corollary 2.4. With reference to the example of Figure 1, Corollary 2.4 yields:

$$\overline{E}(\beta,\alpha_2,s) = \delta_{s+\theta+\frac{\sigma_2}{R}} \otimes \gamma_{R\cdot s,R-\rho_2} \tag{10}$$

i.e., all the equivalent service curves obtained from Theorem 2.1 with $\tau \ge \theta + \sigma_2/R$.

3. SYSTEM MODEL

We analyze a tandem of N *nodes*, connected by links. The tandem is traversed by *flows*, i.e. distinguishable streams of traffic. We are interested in computing a tight end-to-end delay bound for a specific flow, i.e. the *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 minimum service, in the form of a *rate-latency* service curve, with rate R^k and latency θ^k, $1 \le k \le N$. In the above framework, 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. We model a flow as a stream of *fluid*, i.e. we assume that it is feasible to inject and service an arbitrarily small amount of traffic at a node. We assume that flows are constrained by a σ,ρ *leaky-bucket* arrival curve at their ingress node. Leaky-bucket curves are additive, i.e. the aggregate of two leaky-bucket shaped flows is a leaky-bucket shaped flow whose arrival curve is the sum of the two. Hence, without any loss of generality, we assume that at most *one* flow exists along a path and we identify it using the path as a subscript. Based on how the paths of its flows are interleaved, we classify tandems as being either *nested* or *non-nested*. In a *nested* tandem, flows are either *nested* into one another, or they have null intersection. This means that no two flows (i,j), (h,k) exist for which $i < h \le j < k$. Said in other words, let us consider two flows (i,j), (h,k), with $(i,j) \ne (h,k)$ and $i \le h$. Then either $j < h$, or $k \le j$. In the first case, the two flows span a disjoint set of nodes. In the second case, we say that (h,k) is *nested within* (i,j). For example, Figure 3 represents a nested tandem of three nodes. Flow $(3,3)$ is nested within flow $(2,3)$. Furthermore, flows $(1,1)$, $(3,3)$ and $(2,3)$ are nested within $(1,3)$, that is the tagged flow. Given a flow (i,j), we denote its *level of nesting* $l(i,j)$ as the number of flows (h,k) into which it is nested. For instance, with reference to Figure 3, it is $l(1,1) = l(2,3) = 2$, and $l(3,3) = 3$. The level of nesting of the tagged flow is therefore equal to one. The *level of nesting of the tandem* is the maximum level of nesting of one of its flows, which can be easily recognized to be the maximum number of flows crossing a single node. Note that a tandem of N

nodes has a level of nesting no greater than N, and that the maximum number of flows insisting on an N-node nested tandem is $2N-1$.

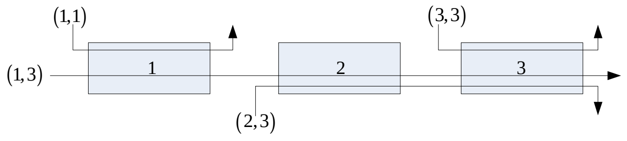


Figure 3 – A nested tandem

A particular case of n-level nested tandem is the one in which $\forall x, 1 \le x \le n, \exists!(i,j): l(i,j) = x$, i.e. we have only *one* flow at each level of nesting. We call such a tandem a *fully nested tandem*. For instance, a sink-tree tandem, i.e. a tandem in which there are exactly N flows, whose path is (i,N), $1 \le i \le N$ (see Figure 4, 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 Figure 4, below. In that case, we say that flow $(1,2)$ and $(2,3)$ are *interdependent*.

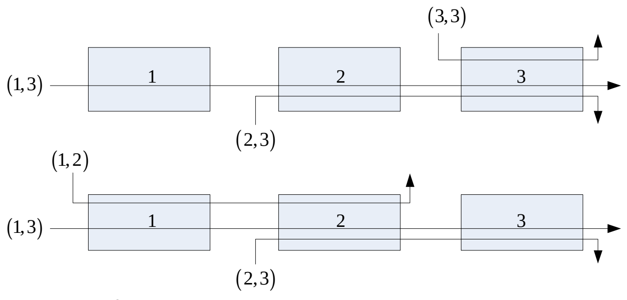


Figure 4 – A fully nested tandem (above) and a non-nested tandem (below)

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 $1 \le h \le N$,

$$\sum_{(i,j): i \le h \le j} \rho_{(i,j)} \le R^h \qquad (11)$$

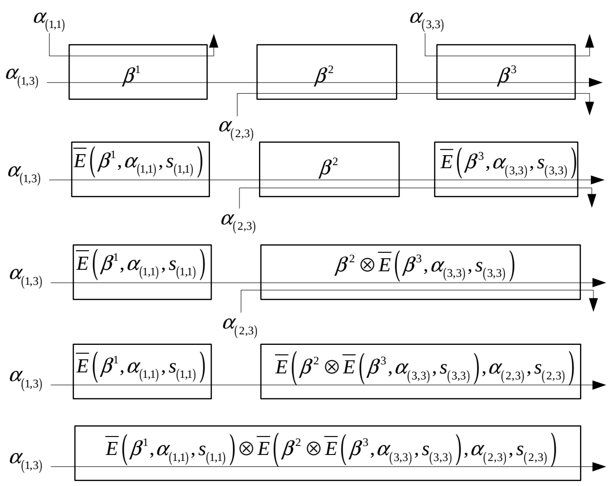
Note that this allows a node's rate to be utilized up to 100%, and it is therefore a necessary condition for stability. Moreover, we assume that the buffer of a node is large enough as to guarantee that traffic is never dropped.

4. THE LEAST UPPER DELAY BOUND METHODOLOGY

In this paragraph, we describe the *Least Upper Delay Bound* (LUDB) methodology and propose numerical techniques for computing delay bounds. We first explain the problem and our solutions for nested tandems, and then extend it to non-nested tandems later on.

At a first level of approximation, LUDB entails computing *all* the service curves for the tagged flow: we start from the aggregate service curves at each node, we apply Corollary 2.4 iteratively in order to remove one flow $(i,j) \ne (1,N)$ from the tandem, and we convolve the service curves of nodes traversed by the same set of flows. Every time Corollary 2.4 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*.

For instance, let us consider again the three-node nested tandem shown in Figure 3. Figure 5 shows how to compute the set of end-to-end service curves for the tagged flow (1,3). We start from the aggregate service curves at each node, and we apply Corollary 2.4, starting from nodes 1 and 3. Then we convolve the service curves obtained for nodes 2 and 3, which are now traversed by the same aggregate of flows (1,3) and (2,3). We remove flow (2,3) by applying once more Corollary 2.4, and we obtain the set of end-to-end service curves for the tagged flow through convolution. The set of service curves $\pi^{\{1,3\}}\left(s_{(1,1)}, s_{(3,3)}, s_{(2,3)}\right)$ depend on three parameters, $s_{(1,1)}$, $s_{(2,3)}$, $s_{(3,3)}$, and they are pseudoaffine for each instance of the three parameters.



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

Figure 5 – An example of application of the LUDB methodology

More generally, let us consider a nested tandem of N nodes, whose level of nesting is $n \ge 2$ (otherwise the problem is trivial). The algorithm for computing the delay bound for the tagged flow can be described as follows.

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:

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

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

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

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 $C_{(h,k)} = \left\{h, h+1, ..., k\right\}$.

For the sake of clarity, hereafter the nodes in the nesting tree are called *t-nodes*, in order to distinguish them to the nodes in the path of the tagged flow. In the nesting tree, there are two kinds of t-nodes: non-leaf t-nodes represent *flows*, and leaf t-nodes represent *sets of nodes* in the path. More specifically:

1. Each non-leaf t-node contains a flow (h,k). The root t-node contains $(1,N)$.
2. Each t-node whose content is (h,k) has all flows $(i,j) \in S_{(h,k)}$ as direct descendants. Furthermore, if $C_{(h,k)} \ne \varnothing$, (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 level of the corresponding t-node in the nesting tree. Accordingly, we henceforth write that $(i,j) \to (h,k)$ iff. $(i,j) \in S_{(h,k)}$, $S_{(h,k)}$ being the set of non-leaf *direct descendants* of (h,k), and that $(i,j) \to^* (h,k)$ to denote that (i,j) is a (possibly non-direct) descendant of (h,k). Figure 6 shows the nesting tree of the tandem of Figure 3. Leaf t-nodes are shown as circles, while non-leaf nodes are ellipses. For instance, it is $(3,3)\rightarrow(2,3)$ and $(3,3)\rightarrow^{*}(1,3)$, whereas $(1,1)\not\rightarrow^{*}(2,3)$.

For non-leaf t-nodes, we also define the *height* $H(i,j)$ as the length of the longest path to a leaf t-node, i.e. $H(2,3)=2$, $H(1,3)=3$. 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:

1. For each *leaf* t-node representing $C_{(h,k)}$ for some parent t-node (h,k), compute
$$\pi^{C_{(h,k)}}=\bigotimes_{j\in C_{(h,k)}}\beta^{j}$$
2. at a non-leaf t-node (h,k), compute a service curve as
$$\pi^{\{h,k\}}=\pi^{C_{(h,k)}}\otimes\left[\bigotimes_{(i,j)\in S_{(h,k)}}\bar{E}\left(\pi^{\{i,j\}},\alpha_{(i,j)},s_{(i,j)}\right)\right]\qquad(12)$$
   i.e. as the convolution of:
   i) The (pseudoaffine) service curves obtained by applying Corollary 2.4 to the service curve computed at all child t-nodes;
   ii) The (rate-latency, hence pseudoaffine) 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 least upper end-to-end delay bound for the tagged flow is the following:
$$V=\min_{\substack{s_{(i,j)}\geq0,\\(i,j)\rightarrow^{*}(1,N)}}\left\{h\left(\alpha_{(1,N)},\pi^{\{1,N\}}\left(s_{(i,j)}:(i,j)\rightarrow^{*}(1,N)\right)\right)\right\}\qquad(13)$$

We now discuss how to to solve (13) numerically. After that, we show how to adapt LUDB to compute delay bounds in non-nested tandems. This section terminates with a wrap-up discussion showing the practicability of the presented algorithms and their improvement over per-node analysis.

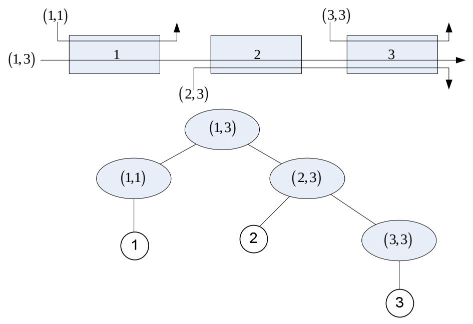


Figure 6 – a nested tandem and the related nesting tree

4.1 Computing the LUDB in nested tandems

Since $\pi^{\{1,N\}}(\;)$ is pseudoaffine and $\alpha_{(1,N)}$ is an affine curve, problem (13) is an optimization problem with a *piecewise linear* objective function of M variables and M linear constraints, M being the number of distinguished *flows* in the tandem (or, equivalently, the number of non-leaf t-nodes in the nesting tree) minus one, $M<2N$. Therefore, (13) is a *piecewise-linear programming* (P-LP) problem. Closed-form solutions have been derived through ad-hoc methods for special cases of problem (13), i.e. for a 2-level nested tandem of arbitrary length [11], i.e. one where one-hop persistent cross flows traverse the tandem, and for a sink-tree tandem [12]. In [14] we showed that problem formulation (13) is the same for all tandems whose nesting trees are *tree-equivalent*, i.e. have the same shape. As a consequence, any closed-form solution computed for a given tandem can be generalized to all tree-equivalent tandems. For instance, the solution for any *fully nested* tandem can be computed in a closed form starting from the one computed for the tree-equivalent sink-tree tandem via simple variable substitution. However, a closed-form solution for a generic nested tandem is still missing as of today.

As far as *numerical* methods are concerned, P-LP problems are normally dealt with either by exploding them into a number of simplexes, by considering each linear piece at a time, or through ad-hoc algorithms that exploit some known properties, e.g., the convexity of the objective function (see, for instance, [10]). Since we have no proof that the objective function is convex, we use a simplex approach to solve (13). Assume for ease of notation that
$$\pi^{\{1,N\}}=\delta_{D}\otimes\left[\bigwedge_{1\leq x\leq n}\gamma_{\sigma_x,\rho_x}\right].$$

By Property 2.3, problem (13) can be formulated as follows:
$$V=\min\left\{D+\bigvee_{1\leq x\leq n}\left[\frac{\sigma_{(1,N)}-\sigma_{x}}{\rho_{x}}\right]^{+}\right\}$$
$$s.t.$$
$$s_{(i,j)}\geq0,\qquad\forall(i,j)\rightarrow^{*}(1,N)$$

Problem (13) has a non 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 curly 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, $1\leq x\leq n$, 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. A generic decomposition x, $1\leq x\leq n$ is formulated as follows:
$$\begin{aligned}&V_{x}=\min\left\{D+\frac{\sigma_{(1,N)}-\sigma_{x}}{\rho_{x}}\right\}\\&s.t.\\&s_{(i,j)}\geq0,\qquad\forall(i,j)\rightarrow^{*}(1,N)\\&\frac{\sigma_{(1,N)}-\sigma_{x}}{\rho_{x}}\geq\frac{\sigma_{(1,N)}-\sigma_{y}}{\rho_{y}}\qquad\forall y,1\leq y\leq n\\&\frac{\sigma_{(1,N)}-\sigma_{x}}{\rho_{x}}\geq0\end{aligned}\qquad(14)$$

While the $n+1$ th one is:
$$\begin{aligned}&V_{n+1}=\min\{D\}\\&s.t.\\&s_{(i,j)}\geq0,\qquad\forall(i,j)\rightarrow^{*}(1,N)\\&\frac{\sigma_{(1,N)}-\sigma_{y}}{\rho_{y}}\leq0\qquad\forall y,1\leq y\leq n\end{aligned}\qquad(15)$$

Then, the LUDB is computed as:

$$V = \min_{1\le x\le n+1}\{V_x\}.$$

Now, if the offset D and the bursts σ_x of the n leaky-bucket stages of $\pi^{\{1,N\}}$ are *affine* functions of $s_{(i,j)} \ge 0,\ (i,j)\to^{*}(1,N)$, then all the decompositions (14)-(15) are simplexes. This happens, for instance, in a tandem with one-hop persistent cross flows, as in [11]. Otherwise, D and σ_x are itself piecewise linear functions of $s_{(i,j)} \ge 0,\ (i,j)\to^{*}(1,N)$. However, through (12) and Property 2.2, 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 simplex originating from a LUDB problem a *recursive simplex decomposition* (RSD) of that LUDB problem. A first solution algorithm for the LUDB problem (13) entails expanding the LUDB formula as explained and solving all the resulting RSDs. We instantiate the RSD algorithm on the nested tandem of Figure 3.

Example 4.1

For the nested tandem of Figure 3 (whose nesting tree is reported in Figure 6) the LUDB problem is the following:

$$\begin{array}{l}
V = \min\left\{\left[\sum_{i=1}^{3}\theta^i + \dfrac{\sigma_{(1,1)}}{R^1} + s_{(1,1)} + \dfrac{\sigma_{(3,3)}}{R^3} + s_{(3,3)} + \Sigma_{(2,3)} + s_{(2,3)}\right] + \Sigma_{(1,3)}\right\}\\
s.t.\\
s_{(1,1)} \ge 0\\
s_{(2,3)} \ge 0\\
s_{(3,3)} \ge 0
\end{array}\qquad (16)$$

where the term between square brackets in (16) is the *offset* of the pseudoaffine service curve for the tagged flow $(1,3)$, and:

$$\Sigma_{(1,3)} = \vee\left[\begin{array}{l}
\dfrac{\sigma_{(1,3)} - \left(R^3 - \rho_{(3,3)}\right)\cdot\left\{s_{(2,3)} + \Sigma_{(2,3)} - \dfrac{\sigma_{(2,3)} - R^3\cdot s_{(3,3)}}{R^3 - \rho_{(3,3)}}\right\}}{R^3 - \rho_{(3,3)} - \rho_{(2,3)}},\\
\dfrac{\sigma_{(1,3)} - R^2\cdot\left\{s_{(2,3)} + \Sigma_{(2,3)} - \dfrac{\sigma_{(2,3)}}{R^2}\right\}}{R^2 - \rho_{(2,3)}},\quad \dfrac{\sigma_{(1,3)} - R^1\cdot s_{(1,1)}}{R^1 - \rho_{(1,1)}}
\end{array}\right]^{+},$$

$$\Sigma_{(2,3)} = \vee\left[\frac{\sigma_{(2,3)} - R^3\cdot s_{(3,3)}}{R^3 - \rho_{(3,3)}}, \frac{\sigma_{(2,3)}}{R^2}\right]^{+}.$$

At a first glance, twelve RSD can be obtained from (16), as many as the cross product of the terms in the two maxima expressions $\Sigma_{(1,3)}$ and $\Sigma_{(2,3)}$, i.e. four and three respectively. Note that, while all the three terms within square brackets in $\Sigma_{(1,3)}$ can be negative, in which case the null term $[\ ]^{+}$ would be the maximum one, the second term in $\Sigma_{(2,3)}$ is non negative by definition, and therefore the null term is not necessary. This reduces by one the number of combinations for $\Sigma_{(2,3)}$ and brings the total number of RSDs to $4\times 2 = 8$.

For the sake of completeness, hereafter we write down the RSD obtained assuming that the maximum is achieved in the *first* term of both $\Sigma_{(1,3)}$ and $\Sigma_{(2,3)}$. Our assumptions yield the following:

$$\Sigma_{(2,3)} = \frac{\sigma_{(2,3)} - R^3\cdot s_{(3,3)}}{R^3 - \rho_{(3,3)}},\quad \Sigma_{(1,3)} = \frac{\sigma_{(1,3)} - \left(R^3 - \rho_{(3,3)}\right)\cdot s_{(2,3)}}{R^3 - \rho_{(3,3)} - \rho_{(2,3)}}$$

And they are obtained under the following inequalities (the first one related to $\Sigma_{(2,3)}$, the following three related to $\Sigma_{(1,3)}$):

$$\begin{array}{l}
\dfrac{\sigma_{(2,3)} - R^3\cdot s_{(3,3)}}{R^3 - \rho_{(3,3)}} \ge \dfrac{\sigma_{(2,3)}}{R^2}\\[2ex]
\dfrac{\sigma_{(1,3)} - \left(R^3 - \rho_{(3,3)}\right)\cdot s_{(2,3)}}{R^3 - \rho_{(3,3)} - \rho_{(2,3)}} \ge \dfrac{\sigma_{(1,3)} - R^2\cdot\left\{s_{(2,3)} + \dfrac{\sigma_{(2,3)} - R^3\cdot s_{(3,3)}}{R^3 - \rho_{(3,3)}} - \dfrac{\sigma_{(2,3)}}{R^2}\right\}}{R^2 - \rho_{(2,3)}}\\[2ex]
\dfrac{\sigma_{(1,3)} - \left(R^3 - \rho_{(3,3)}\right)\cdot s_{(2,3)}}{R^3 - \rho_{(3,3)} - \rho_{(2,3)}} \ge \dfrac{\sigma_{(1,3)} - R^1\cdot s_{(1,1)}}{R^1 - \rho_{(1,1)}}\\[2ex]
\dfrac{\sigma_{(1,3)} - \left(R^3 - \rho_{(3,3)}\right)\cdot s_{(2,3)}}{R^3 - \rho_{(3,3)} - \rho_{(2,3)}} \ge 0
\end{array}$$

Accordingly, after a few straightforward algebraic manipulations we obtain the following RSD:

$$\begin{array}{l}
V_1 = \min\left\{\begin{array}{l}\left[\sum_{i=1}^{3}\theta^i + \dfrac{\sigma_{(1,1)}}{R^1} + s_{(1,1)} + \dfrac{\sigma_{(3,3)}}{R^3} + s_{(3,3)} + \dfrac{\sigma_{(2,3)} - R^3\cdot s_{(3,3)}}{R^3 - \rho_{(3,3)}} + s_{(2,3)}\right]\\ + \dfrac{\sigma_{(1,3)} - \left(R^3 - \rho_{(3,3)}\right)\cdot s_{(2,3)}}{R^3 - \rho_{(3,3)} - \rho_{(2,3)}}\end{array}\right\}\\
s.t.\\
s_{(1,1)} \ge 0\\
s_{(2,3)} \ge 0\\
s_{(3,3)} \ge 0\\
\dfrac{\sigma_{(2,3)} - R^3\cdot s_{(3,3)}}{R^3 - \rho_{(3,3)}} \ge \dfrac{\sigma_{(2,3)}}{R^2}\\[2ex]
\dfrac{\sigma_{(1,3)} - \left(R^3 - \rho_{(3,3)}\right)\cdot s_{(2,3)}}{R^3 - \rho_{(3,3)} - \rho_{(2,3)}} \ge \dfrac{\sigma_{(1,3)} - R^2\cdot\left\{s_{(2,3)} + \dfrac{\sigma_{(2,3)} - R^3\cdot s_{(3,3)}}{R^3 - \rho_{(3,3)}} - \dfrac{\sigma_{(2,3)}}{R^2}\right\}}{R^2 - \rho_{(2,3)}}\\[2ex]
\dfrac{\sigma_{(1,3)} - \left(R^3 - \rho_{(3,3)}\right)\cdot s_{(2,3)}}{R^3 - \rho_{(3,3)} - \rho_{(2,3)}} \ge \dfrac{\sigma_{(1,3)} - R^1\cdot s_{(1,1)}}{R^1 - \rho_{(1,1)}}\\[2ex]
\dfrac{\sigma_{(1,3)} - \left(R^3 - \rho_{(3,3)}\right)\cdot s_{(2,3)}}{R^3 - \rho_{(3,3)} - \rho_{(2,3)}} \ge 0
\end{array}$$

The other seven are obtained through the same procedure, assuming different maxima and related inequalities.

€

In general, each RSD of an LUDB with M cross flows has M variables and a number of constraints C as follows:

$$C = M + L(1,N) \le M + \frac{(M+1)\cdot M}{2} - 1, \qquad (17)$$

where:

$$L(h,k) = \begin{cases} 1 & S_{(h,k)} = \varnothing \\ 1 + \sum\limits_{(i,j)\to(h,k)}\left[L(i,j) - 1_{\{C_{(i,j)} = \varnothing\}}\right] & S_{(h,k)} \ne \varnothing \end{cases} \qquad (18)$$

The first M constraints are simply $s_{(i,j)} \ge 0$ for each cross flow. The other $L(1,N)$ constraints are those required to isolate a single term of the nested sequence of maxima, and they are always no more than the sum of the first M naturals (although our experiments show that their actual number is much smaller on average). Note that $L(h,k)$ is the number of leaky-bucket stages in the pseudoaffine service curve computed at node (h,k).

However, although a simplex with M variables and $O(M^2)$ constraints may look tractable from a computational standpoint in practical cases, the overall *number* of required RSDs may instead grow very fast. In fact, the number Ω of RSDs to be solved can be recursively computed as $\Omega = G_{(1,N)}$, with:

$$G(h,k) = \begin{cases} 1 & S_{(h,k)} = \varnothing \\ L(h,k) \cdot \prod_{(i,j)\to(h,k)} G(i,j) & S_{(h,k)} \neq \varnothing \end{cases} \qquad (19)$$

It is easy to see from (18) and (19) that Ω depends on both the number of flows M and their level of nesting, a deeper nesting tree yielding more RSDs than a shallower one due to the product operator. In fact, Ω ranges between M, achieved in a two-level tandem (i.e. one with one-hop persistent cross-flows), to $M!$, achieved in a sink-tree tandem (although the LUDB can actually be computed in a closed-form in both cases, without actually going through the RSD process).

A much more efficient solution algorithm can be obtained by observing that many of the Ω 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. With reference to the previous example, one can easily check that, if $R^3 - \rho_{(3,3)} \geq R^2$, then the following inequality is infeasible for any $s_{(3,3)} \geq 0$.

$$\frac{\sigma_{(2,3)} - R^3 \cdot s_{(3,3)}}{R^3 - \rho_{(3,3)}} \geq \frac{\sigma_{(2,3)}}{R^2} \qquad (20)$$

As a consequence, all the RSDs which include (20), i.e. four out of eight (included the one described in the above example), are infeasible too, and can be safely skipped. Thus, the key to a faster solution algorithm is to work *bottom-up* in the nesting tree: starting from the t-nodes (i,j) such that $H(i,j)=2$ (e.g., node (2,3) in the above example), 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. Note that this does not guarantee that the resulting RSDs will be feasible themselves, since the subset of constraints causing infeasibility may be sparse among several child t-nodes. However, as soon as new infeasible RSDs are identified, they are marked and ruled out from then on.

The bottom-up algorithm 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 $O(M^2)$. As another example, for a case-study tandem with 30 nodes and 31 flows, nested up to level five, we obtain $\Omega = 1.5 \cdot 10^9$. The bottom-up algorithm brings the overall number of RSDs which have to be solved or proved infeasible (including those at intermediate t-nodes) 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.

It is however evident that, as the scale of the LUDB problem gets larger in both the number of flows and their nesting level, the computation times become impractical also in a bottom-up approach. For this reason, we propose a heuristic method for computing an *approximate* LUDB in larger scale nested tandems.

4.2 Heuristic approximation of the LUDB

When computing the LUDB at a t-node (i,j), each RSD $S_{(i,j)}$ includes a set of constraints. Some of these are related to child t-nodes $(h,k)\to(i,j)$. For instance, with reference to Example 5.1, all the RSDs for the tagged flow (1,3) include one constraint stating which term in $\Sigma_{(2,3)}$ is the largest, with $(2,3)\to(1,3)$. We say that an RSD $S_{(i,j)}$ *includes* another $S_{(h,k)}$, with $(h,k)\to^*(i,j)$, if the set of constraints of $S_{(i,j)}$ includes that of $S_{(h,k)}$. Define a n *optimal* RSD for a t-node (i,j), $S^*_{(i,j)}$, as one whose optimum is the LUDB for (i,j). In general, there can be more than one optimal RSDs for a t-node. Based on experimental observations, it turns out that an optimal RSD often *includes* some optimal RSDs of its children t-nodes $(h,k)\to(i,j)$. In other words, the constraints of optimal RSDs for *children* t-nodes are good candidates for being included in the optimal RSD at a *parent* t-node. With reference again to Example 5.1, if one finds that the optimal RSD at (2,3) is obtained with the following constraint:

$$\frac{\sigma_{(2,3)} - R^3 \cdot s_{(3,3)}}{R^3 - \rho_{(3,3)}} \geq \frac{\sigma_{(2,3)}}{R^2}$$

then the optimal RSD for the tagged flow (1,3) often includes the same constraint.

We capitalize on this in order to devise an effective heuristic technique for approximating the LUDB. The key to efficiency is to *limit* the explosion in the number of the RSDs. In order to do that, we traverse the tree bottom-up, identify the few *best* RSDs for each t-node (i.e., those leading to the best approximate LUDB) and *discard the others*. Going upwards to parent t-nodes, we force the RSD algorithm to keep into account *only* the surviving RSDs of children t-nodes. More specifically, the algorithm is as follows:

1) compute the (exact) LUDB at all t-nodes (i,j) such that $H(i,j)=2$; memorize all the optimal RSDs and discard the others.
2) going bottom-up towards the root t-node, compute an *approximate* LUDB at a t-node (i,j) with $H(i,j)>2$ using the RSD algorithm. However, when the recursion gets to decomposing a term related to a child t-node $(h,k)\to(i,j)$, instead of exploring all possible RSDs, pick *only* the *best* RSDs for that t-node, *up to a maximum of* $n \geq 1$. When more than n are available, select n at random among them. After computing the approximate LUDB at (i,j), memorize all the best RSDs, and discard the others.

The above heuristics allows one to control the number of RSDs which are passed on to a parent t-node, and, as a consequence, the overall computation time. Generally speaking, the smaller n is, the faster an approximated solution is computed, but the less likely it is that the latter is equal to *the* LUDB. However, we show that the trade-off is very favorable, i.e. we can compute very good approximations solving few simplexes. We start with observing that, as far as sink-trees are concerned, the above heuristics always compute an exact solution even with $n=1$. In fact, in this case there is only one feasible simplex at each t-node. In order to evaluate the trade-off between accuracy and computational overhead in more general settings, we create nested tandems whose nesting trees are balanced k-ary trees with a level of nesting equal to l, i.e. having $(k^l - 1)/(k-1)$ flows and k^{l-1} nodes. For each flow, we randomly select a burst and sustainable rate, and for the tandem nodes we select a null latency and a rate equal to:

$$R^h = (1+x) \cdot \sum_{(i,j): i \le h \le j} \rho_{(i,j)} ,$$

where x is a random variable uniformly distributed in $[0.01,1]$, so as to verify (11). The results are plotted as a function of n for a given set of balanced nesting trees, varying their level of nesting and -arity (i.e., l and k). For each selected value of l and k, we generate 50 problems with different data sets.

Figure 7 shows the percentage of scenarios where the heuristic solution matches the LUDB. The figure shows that a relatively small value of n is enough in most cases, the performance worsening when the level of nesting gets higher. More interestingly, even when the heuristics does not achieve the LUDB, the relative error is small to negligible. Figure 8 shows the *maximum* relative error in the LUDB estimate in the same scenarios. Both figures show that, with $n = 5$, you are very likely to attain the LUDB, or a solution less than 1% apart. The heuristics is however considerably faster, as shown in Figure 9. The latter reports the ratio between the number of simplexes solved in the heuristics and those solved for the exact LUDB computation (using the bottom-up approach). The figure shows that the larger the size of the problem gets, the larger the improvement is.

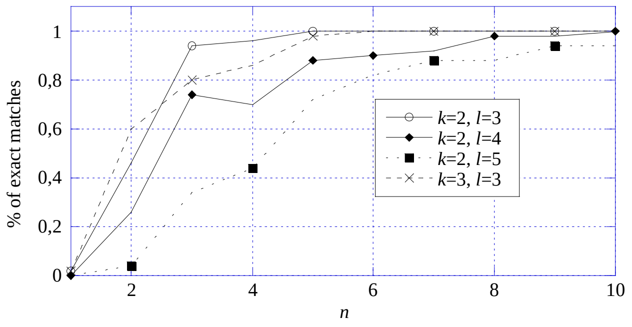


Figure 7 – Percentage of exact matches of the LUDB through the heuristics

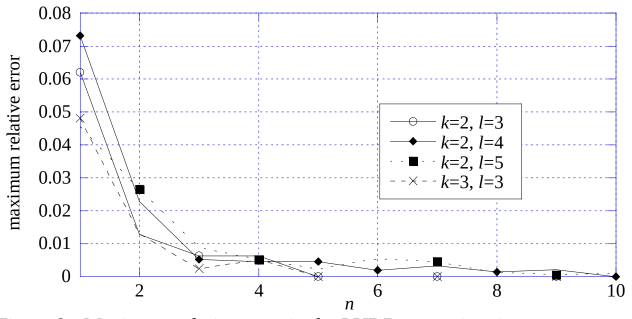


Figure 8 - Maximum relative error in the LUDB approximation

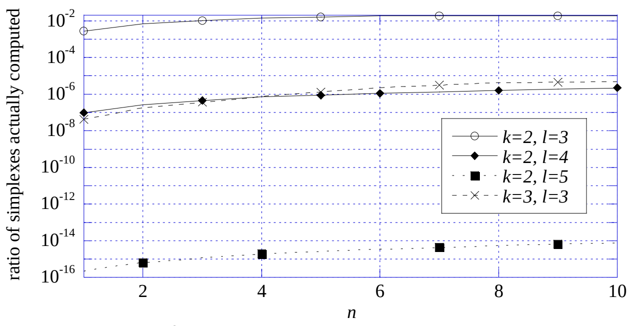


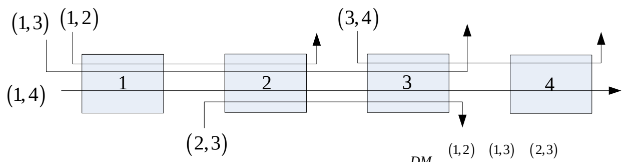
Figure 9 - Fraction of simplexes solved in the heuristics with respect to those required for the exact LUDB computation

In order to give a tangible figure, for the same case study described at the end of the previous subsection, the heuristic solution with $n = 2^{31}$ (i.e., with virtually no limit to the number of combinations carried to the parent level) completes on the same system in 1.6s, returning the same result.

Within the limits of the considered scenarios, the heuristics appear to be very fast and accurate, allowing good estimates of the LUDB to be obtained in reasonable computation times even in large nested tandems. We now move to considering non-nested tandems.

4.3 Non-nested tandems

The LUDB methodology cannot be applied directly to non-nested tandems, such as the one shown in Figure 4. As shown in [14], a non-nested tandem has to be partitioned into a number of nested sub-tandems, each of which can then be analyzed in isolation using LUDB. Whenever two flows (i,j), (h,k) exist for which $i < h \le j < k$, they are said to be *interdependent*, and *cutting* the tandem at (i.e., *before*) any node in $[h, j+1]$ will sever their interdependency. For instance, with reference to Figure 10, flows $(1,2)$ and $(2,3)$ are interdependent, and their interdependency is severed by cutting at any node in $[2,3]$. In order to analyze such tandems, two problems need be solved: first of all, finding *sets of cuts* that partition a tandem so as to sever all the interdependencies (thus creating only nested sub-tandems); then, computing the delay bound for each sub-tandem. With respect to the latter issue, in [14] we have shown that, once a suitable set of cuts is identified, the delay bound can be computed iteratively: starting from the first sub-tandem, we compute the LUDB for the tagged flow *and* the output arrival curve for each flow that crosses the cut (i.e. that is also present in the next sub-tandem); then we move to the next sub-tandem, for which we have just computed all the arrival curves, and so on. We have also shown that computing an output arrival curve implies solving a separate LUDB problem, which is as complex as (or less complex than) the one for delay computation in the sub-tandem. With respect to the first issue, i.e. computing a set of cuts, [14] mentions that there are, in general, many ways to do so for a tandem, as there are always at least *two* nodes to cut at in order to solve a single interdependency. As a consequence, one should – in principle – try *all* possible sets of cuts (optimal ones being hard to identify *a priori*) and select the best delay bound among them *a posteriori*. A fast heuristic is also proposed for computing *one* such set. Hereafter, we delve deeper into both the above problems.



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		×	×

Figure 10 – A sample non-nested tandem, and related flow matrix (left) and dependency matrix (right)

Given a tandem $T=\{i:1\le i\le N\}$ of N nodes, we partition it into m disjoint sub-tandems $T_i=\{k:c_{i-1}\le k\le c_i-1\}$, $1\le i\le m$, with $c_m=N+1$, $c_i<c_{i+1}$ (and $c_0=1$ for ease of notation). c_i is a node index, and is accordingly called a *cut*. Furthermore, we call a *set of cuts* a set of nodes $SC=\{c_i:1\le i\le m\}$ such that $\bigcup_{1\le i\le m}T_i\equiv T$ and $T_i\cap T_j=\varnothing$ for $i\ne j$. Note that the possible sets of cuts for an N-node tandem in the order of the subsets of a set of N elements, i.e. $\Theta(2^N)$. We are interested in *Nesting* SC (*NSC*), i.e. those for which each T_i is a nested sub-tandem. In the above example, $\{3,5\},\{2,4,5\},\{2,3,4,5\}$ are some NSC. However, cutting a tandem entails assuming separate worst-case scenarios for each sub tandem, which are not necessarily possible simultaneously. Therefore, the less often we cut, the tighter the result is going to be. Hence we define a *Primary* SC (*PSC*) as one NSC X such that $\forall j,\ X\setminus\{c_j\}$ is not an NSC. Broadly speaking, PSCs are non-redundant NSCs, and thus the only ones worth considering. With reference to the above example, $a=\{3,5\}$ and $b=\{2,4,5\}$ are PSCs, which originate *two* different end-to-end delay bounds V^a and V^b. The alert reader can check that V^a can be either larger or smaller than V^b, depending on the actual values of the nodes and flows parameters, despite the fact that a has one cut less. Hence the safest choice is to compute $V=V^a\wedge V^b$.

The first problem to solve is therefore how to compute all the PSCs in a tandem. In fact, their number grows with the number of nodes and flows in the tandem. We present an efficient algorithm to do this. In a non-nested tandem, the number of flows M is upper bounded by $N\cdot(N+1)/2$, i.e. all the couples (i,j) with $1\le i\le j\le N$. These flows can be represented in an $N\times N$ binary *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 the following blocks of the matrix:

$$[1,i-1]\times[i,j-1]\cup[i+1,j]\times[j+1,N]$$

Let d be the number of such dependencies. As a first step, an $N\times d$ binary *Dependency Matrix* (*DM*) is computed. For the above example, *F* and *DM* are also shown in Figure 10. 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 matrix has a 1 at all the rows $[h,j+1]$ for that dependency. Building such matrix requires at most $M^2/2$ comparisons. Row n of the DM is thus the set of dependencies D_n severed by a cut at node n. Note that a set of cuts is an NSC if and only if it satisfies all the dependencies in $D=\bigcup_{n=1}^{N}D_n$.

As a second step, all the candidate NSCs are recursively computed as follows (see the pseudocode in Figure 11). First we initialize the following variables: the global list of valid sets of cuts $VSC=\varnothing$, the current set of cuts found $SC=\varnothing$ and the set of severed dependencies $SD=\varnothing$. Then, at each recursive step, a new node $n'\in\{n+1,...,N\}$ is considered for inclusion into SC (we assume $n=1$ initially). The node is discarded if either all the dependencies in $D_{n'}$ are already in SD, or if we are cutting at three consecutive nodes. In this last case, in fact, since any two-node tandem is by definition a nested one, removing one cut would still yield an NSC. On the other hand, if the inclusion test is passed then we update the set of dependencies as $SD'=SD\cup D_{n'}$; now if $SD'=D$, then SC is a valid NSC, hence it is added to VSC and the current recursive iteration terminates. Otherwise a new recursion step is started from the current node and so on, until the last node is reached. The algorithm exits when the top-level iteration terminates, leaving all the possible NSCs stored into VSC.

As a final step, VSC is scanned to eliminate redundant sets of cuts, so as to leave only PSCs. Note that the number of PSCs still grows exponentially with the number of nodes, however with a reduced exponent: for a tandem including *all* possible flows, our experiments show that the number of PSCs grows as $0.73\cdot 2^{0.4N}$, i.e. about $3\cdot 10^3$ for $N=30$.

```
$VSC=\varnothing$; $SC=\varnothing$; $SD=\varnothing$; $n=1$;
void compute_sets ( $SC$, $SD$, $n$ ) {
  for $n'=n+1$ to $N$ {
    if $D_{n'}\subseteq SD$ or $\{n'-2,n'-1\}\subseteq SC$
       then continue; //skip to next node
     $SC'=SC\cup\{n'\}$; //add n to the set of cuts
     $SD'=SD\cup D_{n'}$; //update set of depend.
    if $SD'=D$ then $VSC=VSC\cup SC'$;
    else compute_sets( $SC'$, $SD'$, $n'$ );
  }
}
```

Figure 11 - Pseudo-code for recursive cuts sets computation

Once all the PSCs are computed, we are faced with the problem of computing the related delay bounds. As already observed, one delay bound should be computed per PSC, using repeated LUDB computation. To this aim, one may note that cutting a long tandem into shorter sub-tandems actually reduces the overall complexity: in other words, computing the delay bound for a PSC entails solving a linear number of exponentially simpler problems, which is generally much faster. However, this would obviously be computationally expensive as the number of PSCs grows large. In order to obtain an efficient implementation, we can again exploit the problem structure. Since an interdependency between two flows can be severed by cutting *at least* at two different nodes, there will be in general several PSCs where the first n cuts are the same. For these, the delay bounds and output arrival curves in the first n sub-tandems can be computed once, and reused in the subsequent computations. This can easily be done by arranging all the PSCs in a tree, whose $(j+1)^{th}$ level nodes are the j^{th} cuts in every PSC. The end-to-end delay bounds for all the PSCs can be efficiently computed by visiting such a tree depth-first and saving all the computations at intermediate nodes. Table 1 reports the percentage of LUDB skipped with respect to a brute-force approach, each value representing an average over ten randomly generated instances. The number of flows is varied among 20%, 50% and 100% of all the possible ones. The flow paths are generated at random (the resulting tandem is checked to be non-nested before proceeding with the analysis), and so are their rates and bursts. Nodes are provisioned so as to accommodate the overall flow traversing them, with a random overprovisioning ranging from 1% to 50%, while their latencies are set to be identically null. As the table clearly shows, the amount of computations saved is significant, increasing with both the number of nodes and the flow density.

N, *d*	**20%**	**50%**	**100%**
10	20.42	39.33	41.67
15	45.36	56.02	58.77
20	61.40	65.59	68.58
25	70.51	73.36	74.74
30	75.47	77.90	78.90

Table 1 – % of LUDBs skipped with respect to a brute-force approach

To evaluate the overall scalability of the proposed approach, we run the computations on non-nested tandems of varying size, in the same settings as before and on the same hardware. Figure 12 shows the time taken to compute all the PSCs against the tandem size. The overall LUDB computation time once the sets of PSCs are identified is instead reported in Figure 13. Both figures show an exponential dependency with respect to the tandem size. Interestingly, the PSC computation time does not depend monotonically on the number of flows, the highest one being achieved for a 20%. This is because, when flows are sparser, sets D_n tend to have smaller intersections, so that you need more to compute an NSC, and therefore the number of their combinations increases as well. Although not shown in the figure, however, further reducing the number of flows (e.g., to 10%) eventually leads to smaller computation times. On the other hand, the LUDB computation gets heavier as the number of flows increases, which is expectable, and represents the dominant time with densely populated tandems. However, as the figures show, a 30-node tandem with all possible flows (i.e., 465) can be analyzed in twenty minutes.

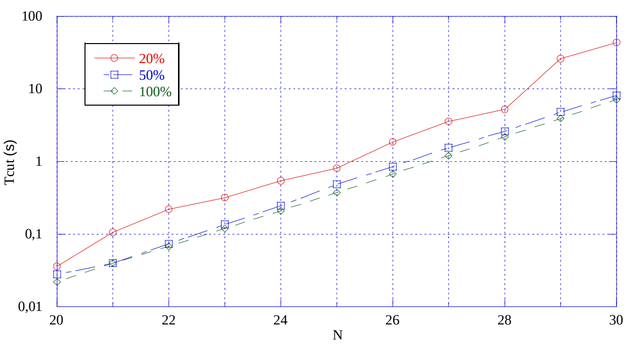


Figure 12 – PSC computation time

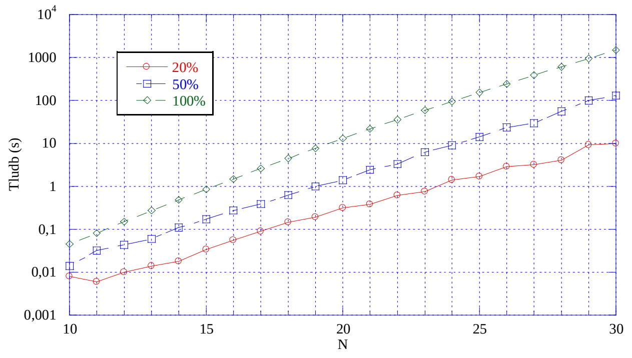


Figure 13 – LUDB computation time

Note that – unless very large nested sub-tandems are obtained (i.e., in the order of several tens of nodes, which is note the case in the above experiments) – heuristics such as those described in Section 4.2 are ineffective in reducing the overall computation time. In fact, when sub-tandems are short, few simplexes are required to solve their LUDB problems, and the efficiency gain of using the heuristics is thus negligible.

4.4 A wrap-up discussion

As shown in the previous sections, LUDB computation completes in seconds or minutes for *nested* tandems of up tens of nodes. Computation in non-nested tandems instead requires minutes for tandems of up to 30 nodes. Since the computation time grows exponentially, and although using more performing hardware, such as quad-core PCs, may push the limit a few units ahead, exceeding this last figure quickly leads to unfeasibly long computation times. However, we observe that 30-node paths are rare and close to the longest in today's planetary Internet [26]. More to the point, few, if any, paths under the control of a *single* administrative entity are that long. This means that such methodology is suitable for today's Internet. Moreover, according to the current trends, paths in the Internet are getting shorter over time due to increasing domain-level interconnectivity, which makes that figure large enough also for the foreseeable future. The point that we want to make now is that such computational overhead is really needed, since it comes with a dramatic improvement in the solution accuracy with respect to the only other comparable method so far, i.e. per-node delay analysis. In the latter, per-node delay bounds and output arrival constraints are computed and summed up, similarly to what we would do by using a (redundant) NSC $X = \{1, ..., N+1\}$. This method is computationally simpler, since single-node analysis is trivial. However, the ratio of the per-node delay bound and the LUDB is always greater than one, and it grows *exponentially* with the number of nodes, in both nested and non-nested tandems. Figure 14 shows the ratio of per-node delay bound over the LUDB as a function of the number of nodes, in the same non-nested tandem settings previously used. The improvement grows from 4-6 times (for 10 nodes) to 500-800 times (for 30 nodes), almost irrespective of the flow density. For nested tandem the gain is even larger: for instance, in a balanced tree with $l = 4, k = 3$ (i.e., 64 nodes), the *heuristic* bound is on average $94 \cdot 10^3$ times smaller than the per-node one.

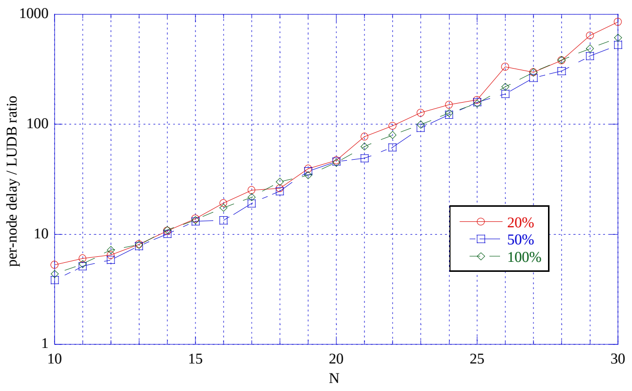


Figure 14 - Ratio between per-node delay and LUDB

Having shown that LUDB analysis is indeed worth pursuing if delay bound accuracy is a concern, we take over the problem of assessing how *tight* the LUDB is in the next section.

5. THE PROBLEM OF TIGHTNESS

Assessing whether the LUDB is a tight bound is made particularly challenging by the fact that a method for computing the WCD in FIFO tandems is still missing. In a previous work of ours, [12], LUDB was applied to sink-tree tandems, which are in fact nested tandems. For this class of tandems, we showed that the LUDB (which can be computed in a closed form) is actually *equal* to the WCD. The proof was obtained by constructing a scenario where a bit of the tagged flow experiences a delay equal to the LUDB itself.

As far as non-nested tandems are concerned, we already showed that this method yields much better results compared to per-node analysis. However, breaking the end-to-end analysis, i.e. computing and summing *partial* delay bounds, is likely to lead to loose end-to-end delay bounds nonetheless, as it entails assuming independent, non simultaneously possible worst-case scenarios at each sub-tandem. The alert reader will notice that a similar argument has been used in the past to prove that the *pay burst only once* principle holds for single flows in per-flow scheduling networks (see e.g. [5] for some discussion on this topic). Broadly speaking, cutting a tandem into sub-tandems is much better than cutting it into single nodes (as it is done in per-node analysis), but mostly because you need *less* cuts to accomplish the same task, hence we would not expect the bounds thus obtained to be tight. If we give for granted that end-to-end analysis is a *necessary* condition to obtain tight bounds, a natural question is whether the latter is also *sufficient*. In other words, whether end-to-end analysis always yields a bound which is equal to the WCD. In this section, we show by counterexample that this is not the case: the LUDB may be larger than the WCD, even when end-to-end analysis is possible. We show that there are cases of nested tandems where we are able to compute a *smaller* delay bound than the LUDB, which proves that the LUDB itself is not necessarily equal to the WCD.

Our line of reasoning is the following: instead of looking for another *method* to upper bound the WCD in a sample tandem (which, to the best of our knowledge, has not be discovered so far), we look for another *tandem*, derived from the sample one, to which to apply the *same method*, i.e., the LUDB. More specifically, consider a tandem T, and call W its WCD. Assume you are able to build a tandem $\overline{T}$, such that its WCD $\overline{W}$ is *not smaller than* W, i.e. $\overline{W} \geq W$. Now, the LUDB is not smaller than the WCD by definition. Thus, if V and $\overline{V}$ are LUDBs for T and $\overline{T}$, then it is $\overline{V} \geq W$, i.e. $\overline{V}$ is obviously a delay bound for T. However, if we find cases when $\overline{V} < V$, we can prove that $V > W$.

Apparently, we need a method to build such a tandem $\overline{T}$ from T. The starting point for this is a technical Lemma which identifies assumptions on the nodes behavior which are compatible with the worst-case scenario. Hereafter, we denote as $A_{(i,j)}^{k}(t)$ the CAF for flow (i,j) at node k, and with $D_{(i,j)}^{k}(t)$ the CDF for flow (i,j) at node k. Furthermore, we denote with $A^{k}(t)$ and $D^{k}(t)$ the *total* CAF and CDF at node k.

Define a *scenario* g for an N-node tandem as:

1) a set of CAFs for all the flows $(i,j) \subseteq (1,N)$ at their entry node, $A_{(i,j)}^{i}(t)$;
2) a set of "node behaviors", i.e. the way each node i, $1 \leq i \leq N$, transforms its CAF $A^{i}(t)$ into its CDF $D^{i}(t)$, according to the related service curve inequality $D^{i}(t) \geq \left[A^{i} \otimes \beta^{i}\right](t)$. As for the latter, we can describe a node behavior by means of a non-negative *lead function* $L^{i}(t)$, which is such that $D^{i}(t) = \left[A^{i} \otimes \beta^{i}\right](t) + L^{i}(t)$. Note that $L^{i}(t)$ is not necessarily wide-sense increasing.

In order for a scenario to be *feasible*, each CAF has to be compatible with the related arrival curve constraint, $A_{(i,j)}^{i}(t) - A_{(i,j)}^{i}(s) \leq \alpha_{(i,j)}(t-s)$. Furthermore, each lead function has to verify $L^{i}(t) \leq A^{i}(t) - \left[A^{i} \otimes \beta^{i}\right](t)$ in order for node i to have a causal behavior.

We first show that there is at least one scenario where all nodes are *lazy*, i.e. they have a null lead, where the WCD is attained.

Lemma 5.1

Assume that a tandem of N FIFO nodes is traversed by a set of flows, and fix the CAF of each flow (i,j) at its entry node, $A_{(i,j)}^{i}$. Then, the WCD for flow $(1,N)$ is achieved in a scenario where all nodes are lazy.

Proof

Call Γ the set of all feasible scenarios in a tandem. Throughout this proof, we express the fact that a quantity depends on scenario $g \in \Gamma$ by using the conditional notation $\big|_{g}$, i.e. $A_{(i,j)}^{i}(t)\big|_{g}$ denotes the CAF of flow (i,j) at node i under scenario g.

Call $d^{i}(t)\big|_{g}$ the delay experienced at node i by a bit of the tagged flow entering a generic N-node tandem at time t in scenario g. The WCD d is defined as follows:

$$d = \max_{g \in \Gamma} \left\{ \max_{t \geq 0} \left[\sum_{i=1}^{N} d^{i}(t)\big|_{g} \right] \right\} \tag{21}$$

This said, we prove the thesis by induction on the nodes, starting from the last one.

Base step: node N has to be lazy.

Call $\Phi_N \subset \Gamma$ the subset of scenarios where $L^{N}(t) = 0$, i.e. those for which node N is *lazy*. We show that at least one worst-case scenario is included in Φ_N, i.e.:

$$d = \max_{g \in \Phi_N} \left\{ \max_{t \geq 0} \left[\sum_{i=1}^{N} d^{i}(t)\big|_{g} \right] \right\} \tag{22}$$

Assume by contradiction that:

$$d > \max_{g \in \Phi_N} \left\{ \max_{t \geq 0} \left[\sum_{i=1}^{N} d^{i}(t)\big|_{g} \right] \right\} \tag{23}$$

and call $x \in \Gamma \setminus \Phi$ the scenario where d is achieved. Consider now the scenario $y \in \Phi_N$, which only differs from x because

$L^N(t)=0$. It is obviously $d^i(t)\big|_y = d^i(t)\big|_x$, $1\le i\le N-1$, since nothing has changed at the first $N-1$ nodes, and $A^N(t)\big|_y = A^N(t)\big|_x$. However, if node N is lazy in y and not in x, it is $D^N(t)\big|_y \le D^N(t)\big|_x$, hence $D^N_{(1,N)}(t)\big|_y \le D^N_{(1,N)}(t)\big|_x$ since the node is FIFO, and $d^N(t)\big|_y \ge d^N(t)\big|_x$. Thus, we have found a scenario $y\in\Phi_N$ where a delay larger than or equal to d is achieved, which contradicts (23).

Inductive step:

Let N be a lazy node. Fix its arrivals $A^N_{(N,N)}(t)$, if any. Then, given a generic scenario $y\in\Phi_N$, we have:

$$D^N(t)\big|_y = \left[X^{N-1}(t)\big|_y + A^N_{(N,N)}(t)\big|_y\right]\otimes\beta^N(t)$$

Where $X^{N-1}(t)\big|_y$ is the sum of the CDFs at node $N-1$ of the flows traversing both $N-1$ and N. Consider now the scenario $x\in\Phi_N$, which only differs from y by assuming that node $N-1$ is lazy as well. We readily obtain that $D^{N-1}(t)\big|_x \le D^{N-1}(t)\big|_y$, which also implies that $X^{N-1}(t)\big|_x \le X^{N-1}(t)\big|_y$ due to the FIFO hypothesis. Since convolution is isotonic, it is then

$$\begin{aligned} D^N(t)\big|_x &= \left[X^{N-1}(t)\big|_x + A^N_{(N,N)}(t)\big|_x\right]\otimes\beta^N(t) \\ &\le \left[X^{N-1}(t)\big|_y + A^N_{(N,N)}(t)\big|_y\right]\otimes\beta^N(t) = D^N(t)\big|_y \end{aligned} \tag{24}$$

For the FIFO hypothesis, (24) implies that $D^N_{(1,N)}(t)\big|_x \le D^N_{(1,N)}(t)\big|_y$. Thus, the horizontal distance between any point in $A^1_{(1,N)}$ and $D^N_{(1,N)}$, i.e. the end-to-end delay of each bit for flow $(1,N)$, is larger or equal if node $N-1$ is lazy. This means that there exists a worst-case scenario in which node $N-1$ is lazy. By repeating the same argument at nodes $N-j$, $2\le j\le N-1$, the thesis follows.

€

Although the above lemma is not sufficient to identify a possible worst-case scenario, it can be used to state the property that allows us to build a tandem $\overline{T}$ from a given tandem T, which we call *Flow Extension (FE)*. We first formulate and prove it, and then exploit it to construct simple counterexamples.

Theorem 5.2 (Flow Extension, FE)

Let T be a tandem of N nodes, in which there is a flow $(j,N-1)$. Call $\overline{T}$ the tandem obtained from T by "extending" flow $(j,N-1)$, i.e. by substituting it with flow (j,N), all else being equal. Call d and $\overline{d}$ the WCD for the tagged flow in T and $\overline{T}$. Then, it is $\overline{d}\ge d$.

Proof

By Lemma 5.1, the WCD is attained in a scenario where all nodes are lazy. Thus, we compare T and $\overline{T}$, limiting ourselves to the subset of feasible scenarios Φ where all nodes are lazy. Whenever needed, we use the same symbol to denote the same quantities in T and $\overline{T}$, adding a bar to the latter ones in order to distinguish them. Consider now a generic scenario $g\in\Phi$ for tandem T, and define the *corresponding* scenario $\overline{g}$ in $\overline{T}$ as the one with the same set of CAFs at the entry nodes of all flows. Clearly, if the scenario is feasible in T, it is also feasible in $\overline{T}$, since flows are subject to the same constraints. However, in tandem $\overline{T}$, flow $(j,N-1)$ is *extended* up to node N. This is exactly like adding a "virtual" flow $(N-1,N)$, with $A^N_{(N-1,N)}(t)=D^{N-1}_{(j,N-1)}(t)$, as an input to node N.

For a scenario $g\in\Phi$ in T, the corresponding scenario $\overline{g}\in\overline{\Phi}$ is such that:

$$\overline{d^i}(t)\big|_{\overline{g}} \ge d^i(t)\big|_g,\ 1\le i\le N. \tag{25}$$

In fact, equality holds in (25) for $1\le i\le N-1$, since the two scenarios are the same up to node $N-1$ included. However, the input at node N in $\overline{T}$ is $\overline{A^N}(t)=A^N(t)+A^N_{(N-1,N)}(t)$, where $A^N_{(N-1,N)}(t)$ is a wide-sense increasing function. Now, since node N is lazy and FIFO, the delay of each bit in $A^N(t)$ cannot be lower than in T, thus $\overline{d^N}(t)\big|_{\overline{g}} \ge d^N(t)\big|_g$.

Now, for any scenario $g\in\Phi$ there exists a scenario $\overline{g}\in\overline{\Phi}$ where the end-to-end delay of a bit of the tagged flow entering at time t in tandem $\overline{T}$ is larger than (or equal to) the one in tandem T. Therefore, the same inequality also holds between the respective WCDs, i.e. $\overline{d}\ge d$.

€

We now show how to exploit FE to compute *smaller* bounds than the LUDB.

Example 5.3

Consider the two-node tandem T shown in Figure 15, left.

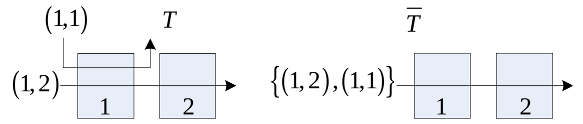


Figure 15 – Two simple tandems. The one on the right is obtained by applying FE to the one on the left.

Build the corresponding tandem $\overline{T}$ according to FE (shown in the same figure on the right), for which it is $\overline{W}\ge W$. Consider now what *delay bound* we can compute through LUDB in both tandems. In T, it is the following:

$$V=\begin{cases} \sum_{i=1}^2\theta^i+\dfrac{\sigma_{(1,1)}}{R^1}+\dfrac{\sigma_{(1,2)}}{R^2} & R^2+\rho_{(1,1)}<R^1 \\ \sum_{i=1}^2\theta^i+\dfrac{\sigma_{(1,1)}}{R^1}+\dfrac{\sigma_{(1,2)}}{R^1\cdot\dfrac{R^2}{R^2+\rho_{(1,1)}}} & R^2+\rho_{(1,1)}\ge R^1 \end{cases} \tag{26}$$

provided that the following provisioning inequalities hold:

$$R^1\ge\rho_{(1,1)}+\rho_{(1,2)},\ R^2\ge\rho_{(1,2)}, \tag{27}$$

otherwise it is infinite.

On the other hand, the LUDB for the tagged flow in $\overline{T}$ is:

$$\overline{V}=\sum_{i=1}^2\theta^i+\frac{\sigma_{(1,1)}+\sigma_{(1,2)}}{R^1\wedge R^2}, \tag{28}$$

provided that the following provisioning inequalities holds:

$$R^1\ge\rho_{(1,1)}+\rho_{(1,2)},\ R^2\ge\rho_{(1,1)}+\rho_{(1,2)}, \tag{29}$$

otherwise it is infinite. Note that the second inequality in (29), related to node 2, is more constraining than the corresponding one in (27).

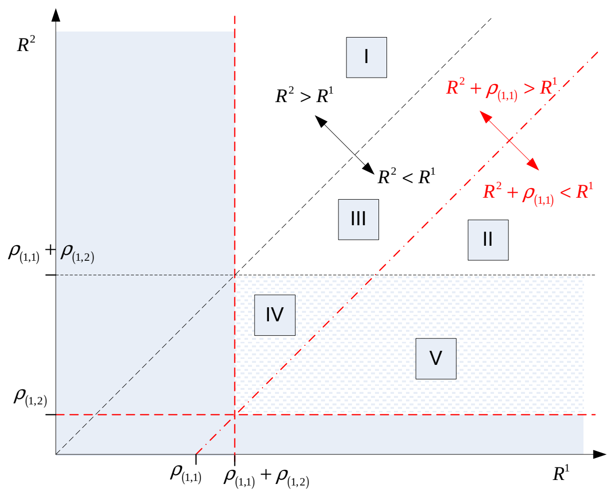
Now, $\wedge(V,\overline{V})$ is a delay bound in T. However, it is easy to see that $\overline{V}<V$ in some cases. Table 2 reports the comparison between V and $\overline{V}$ in the five different regions in which the rate inequalities included in expressions (26)-(29) divide the plan R^1OR^2 (also shown in Figure 16). In region I, $\overline{V}<V$. Thus, the following set of inequalities hold:

$$V\ge W,\quad \overline{V}\ge\overline{W},\quad \overline{W}\ge W,\quad \overline{V}<V \tag{30}$$

An immediate consequence of (30) is that $V>W$, i.e. *the LUDB is not the WCD in that case*.

Furthermore, note that in region III, the rate inequalities are not sufficient to decide whether $\overline{V}<V$ or $\overline{V}\ge V$: in fact, both can occur depending on the values of the parameters. Again, this means that the LUDB is not necessarily the WCD in that region too.

€



R^2
I
$R^2 > R^1$
$R^2 + \rho_{(1,1)} > R^1$
$R^2 < R^1$
$R^2 + \rho_{(1,1)} < R^1$
III
II
$\rho_{(1,1)} + \rho_{(1,2)}$
IV
V
$\rho_{(1,2)}$
$\rho_{(1,1)}$
$\rho_{(1,1)} + \rho_{(1,2)}$
R^1

Figure 16 - Different regions of the plan R^1OR^2 and related inequalities.

Now, when LUDB is applied to a nested tandem, the entire set of all the "good" end-to-end service curves that can be computed using Theorem 2.1 and convolution is explored, and a global minimum is computed. This means that no better bounds can be computed by relying on Theorem 2.1 alone. However, this is proved not to be sufficient for computing the WCD. A likely cause for this is that not all the necessary information is retained in the equivalent service curves computed through Theorem 2.1.

Consider, for instance, a single rate-latency node traversed by two leaky-bucket shaped flows, as in the example shown in Figure 1, and assume that the arrival curve of the two flows are $\alpha_i = \gamma_{\sigma_i,\rho_i}$, $1 \le i \le 2$. The LUDB for flow 1 is computed as the solution of the following trivial optimization problem:

$$d = \min_{s \ge 0}\left\{ s + \theta + \frac{\sigma_2}{R} + \left[\frac{\sigma_1 - R\cdot s}{R - \rho_2}\right]^+ \right\}$$

The minimum is achieved when $s = \sigma_1/R$, and it is equal to $V = \theta + (\sigma_1 + \sigma_2)/R$. This is also the WCD for flow 1, since it is attained by its σ_1 th bit in the following worst-case scenario:

a) both flows are *greedy:* $A_i(t) = \alpha_i(t)$, i.e. their CAFs are equal to their respective arrival curves. However, the burst of flow 2 arrives *just before* that of flow 1.

b) the node is lazy.

Call $D_1(t)$ the CDF for flow 1 obtained in the above scenario, shown in Figure 17 as a thicker dashed line. Let us compare it to the curves $D_1^{'}(t,s)$ obtained by convolving the greedy CAF of flow 1 with each equivalent service curve derived through Corollary 2.4, therein including the "optimum" one. These are shown as thinner lines in the same figure, for various values of s, and they represent *lower bounds* to any CDF that can be obtained from that CAF, by definition of (equivalent) service curve. However, one can easily see that $\nexists s : D_1(t) = D_1^{'}(t,s)$. More to the point, the curves $D_1^{'}(t,s)$ with $s > 0$ cannot be obtained in a FIFO system, since they assume that flow 1 does not transmit any bit for longer than $\theta + \sigma_2/R$. This seems to suggest that the $D_1^{'}(t,s)$ might not be tight lower bounds themselves. This, in turn, would imply that each equivalent service curve alone cannot describe the behavior of a FIFO node with the necessary accuracy.

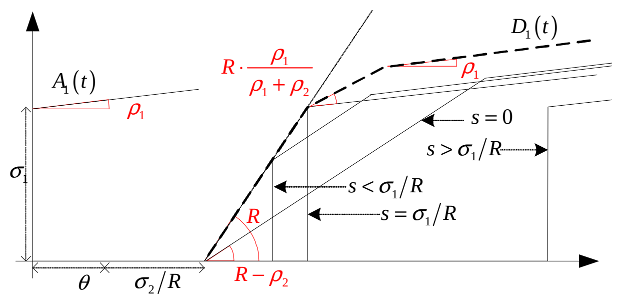


Figure 17 – CDFs obtained using equivalent service curves

5.1 Practical Applications of Flow Extension

Beside being useful to prove the limitations of Theorem 2.1, FE can also be exploited to compute improved delay bounds. However, its practical usefulness is limited for at least three reasons. The first one is represented by the topology restrictions required in

Region	Rate Inequalities	V	$\overline{V}$	Comparison
I	$R^1 \ge \rho_{(1,1)} + \rho_{(1,2)}$, $R^2 > R^1$	$\sum_{i=1}^2 \theta^i + \frac{\sigma_{(1,1)}}{R^1} + \frac{\sigma_{(1,2)}}{R^1\cdot\frac{R^2}{R^2+\rho_{(1,1)}}}$	$\sum_{i=1}^2 \theta^i + \frac{\sigma_{(1,1)}+\sigma_{(1,2)}}{R^1}$	$\overline{V} < V$
II	$R^2 \ge \rho_{(1,1)} + \rho_{(1,2)}$, $R^2 + \rho_{(1,1)} < R^1$	$\sum_{i=1}^2 \theta^i + \frac{\sigma_{(1,1)}}{R^1} + \frac{\sigma_{(1,2)}}{R^2}$	$\sum_{i=1}^2 \theta^i + \frac{\sigma_{(1,1)}+\sigma_{(1,2)}}{R^2}$	$V \le \overline{V}$
III	$R^2 \ge \rho_{(1,1)} + \rho_{(1,2)}$, $R^2 < R^1$, $R^2 + \rho_{(1,1)} \ge R^1$	$\sum_{i=1}^2 \theta^i + \frac{\sigma_{(1,1)}}{R^1} + \frac{\sigma_{(1,2)}}{R^1\cdot\frac{R^2}{R^2+\rho_{(1,1)}}}$	$\sum_{i=1}^2 \theta^i + \frac{\sigma_{(1,1)}+\sigma_{(1,2)}}{R^2}$	It depends
IV	$R^1 \ge \rho_{(1,1)} + \rho_{(1,2)}$, $R^2 < \rho_{(1,1)} + \rho_{(1,2)}$, $R^2 + \rho_{(1,1)} \ge R^1$	$\sum_{i=1}^2 \theta^i + \frac{\sigma_{(1,1)}}{R^1} + \frac{\sigma_{(1,2)}}{R^1\cdot\frac{R^2}{R^2+\rho_{(1,1)}}}$	∞	$V < \overline{V}$
V	$R^2 \ge \rho_{(1,2)}$, $R^2 < \rho_{(1,1)} + \rho_{(1,2)}$ $R^2 + \rho_{(1,1)} \ge R^1$	$\sum_{i=1}^2 \theta^i + \frac{\sigma_{(1,1)}}{R^1} + \frac{\sigma_{(1,2)}}{R^2}$	∞	$V < \overline{V}$

order to apply Theorem 5.2 (i.e., that there is a flow in the tandem that leaves at node $N-1$). Second, in order for it to be of any practical use, it requires that the last node be *overprovisioned*. In fact, with reference to the previous example, we can observe that, if $R^2 \in \left[\rho_{(1,2)}, \rho_{(1,1)} + \rho_{(1,2)}\right[$, i.e. in regions IV and V, the WCD in tandem $\overline{T}$ is infinite, and thus FE is useless in this case. The third one is that it cannot be stated *a priori* whether it will yield smaller bounds or not, as it depends on the actual values of the nodes' and flows' parameters.

This said, we can still find some useful generalization of Theorem 5.2. The first one is that, given a tandem T, and a set of *extensible* flows $S \equiv \left\{(j, N-1) \subset (1, N)\right\}$, FE can in fact be applied by extending any (non empty) *subset* of flows $s \subseteq S$. Thus we can build up to $2^{|S|}-1$ different tandems $\overline{T}$, for each one of which a delay bound can be computed, possibly improving on the LUDB in T for some value of the nodes and flows parameters. However, the more flows are in s, the more constraining the provisioning inequalities at node N must be, in order for the related bound in $\overline{T}$ to be finite. More specifically, the required inequality is the following:

$$\sum_{(i,N)\subset(1,N)} \rho_{(i,N)} + \sum_{(i,N-1)\in s} \rho_{(i,N-1)} \le R^N \tag{31}$$

Thus, the amount of overprovisioning at node N may act as a constraint on the number of *effective* ways in which FE can be applied (which can therefore be smaller than $2^{|S|}-1$ in practice).

The second generalization is that FE can be applied *more than once* to the same tandem, while obviously tightening the provisioning inequalities each time. For instance, in the tandem shown in Figure 18, above, FE can be applied a first time by extending flow $(1,2)$. After convolving the service curves of node 2 and 3, it can then be applied again, extending flow $(1,1)$ up to node 3. One must observe, however, that whether applying FE yields a smaller LUDB or not cannot be decided *a priori*, since it depends on the actual numbers (i.e., node rates, flows bursts, etc.).

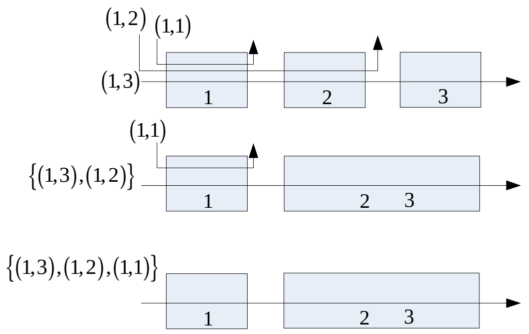


Figure 18 – a three-node nested tandem and related FE transformations

Hereafter, we report another example for FE, this time related to a non-nested tandem.

Example 5.4

Consider the non-nested tandem of Figure 4. We apply FE to it, by extending flow $(1,2)$, and derive the following delay bound:

If $R^1 + \rho_{(2,3)} < \left(R^2 \wedge R^3\right)$, then:

$$\overline{V} = \sum_{i=1}^{3} \theta^i + \frac{\sigma_{(2,3)}}{R^2 \wedge R^3} + \frac{\sigma_{(1,3)} + \sigma_{(1,2)}}{R^1} \tag{32}$$

Otherwise

$$\overline{V} = \sum_{i=1}^{3} \theta^i + \frac{\sigma_{(2,3)}}{R^2 \wedge R^3} + \frac{\sigma_{(1,3)} + \sigma_{(1,2)}}{\left(R^2 \wedge R^3\right) \cdot \dfrac{R^1}{R^1 + \rho_{(2,3)}}} \tag{33}$$

Both (32) and (33) hold provided that $R^3 \ge \rho_{(1,3)} + \rho_{(2,3)} + \rho_{(1,2)}$. Two different delay bound expressions V^a and V^b can be obtained using LUDB in the same tandem, using either $\{3,4\}$ or $\{2,4\}$ as a set of cuts respectively. They are reported in the Appendix. The alert reader can check that, unlike in V^a and V^b, in $\overline{V}$ each burst $\sigma_{(i,j)}$ appears exactly once. It is easy to identify regions in which $\overline{V} < \left(V^a \wedge V^b\right)$. For instance, if $R^i = 3$, $\theta^i = 1$, $1 \le i \le 3$, and $\sigma_{(i,j)} = 3$, $\rho_{(i,j)} = 1$, for all flows, we obtain $\overline{V} = 20/3$, $V^a = 101/9$, $V^b = 92/9$, so that $\overline{V} \simeq 0.65 \cdot \left(V^a \wedge V^b\right)$.
€

6. A LOWER BOUND ON THE WORST-CASE DELAY

Once acquired that the LUDB (whether FE is employed or not) might be an overrated estimate of the WCD, we need a method to assess its tightness. In order to assess how tight an upped bound V is, we compute a *lower* bound v on the worst-case delay. The interval $[v, V]$ includes the WCD by definition, hence we define $U = 1 - v/V$ as the *Relative Overrating Bound* (ROB), meaning that V is overrated by less than a factor U.

Now, any *attainable* end-to-end delay is by definition a lower bound on the worst-case delay, the latter being in fact the maximum attainable delay. Therefore, we compute v by heuristically defining a scenario, i.e. by tuning the arrivals at each flow and assuming a behavior at each node, so that the tagged flow experiences a "large" end-to-end delay. In such a scenario, we inject traffic from both the tagged flow and the cross flows, and we compute how the CAF of the tagged flow is transformed at each node because of FIFO multiplexing and aggregate scheduling into rate-latency service curve elements. Thus, we ultimately compute the CDF for the tagged flow at node N of the tandem, and compute the lower bound as the maximum horizontal distance between the CAF of the tagged flow at node 1 and the CDF at node N.

The idea of using a lower bound in order to assess the tightness of a Network Calculus upper bound under FIFO multiplexing has already been used in [21], in the context of sink-tree networks. Before introducing the scenario that we use to evaluate the lower bound, we need to describe the algorithmic framework that we use to manipulate CAFs at each node under FIFO multiplexing, showing how to compute per-flow CDFs.

6.1 An Algorithmic Framework for Network Calculus with FIFO-multiplexing Nodes

Within the DEBORAH tool, we represent each CAF as a piecewise linear function, without any hypothesis on convexity. While this allows us to approximate any curve using a suitably large amount of segments, the curves that we will actually use in our scenario are piecewise linear, so their representation is exact. Therefore, each CAF $A_{(i,j)}^k$ is a list of $Q_{(i,j)}^k$ 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: $B_i = \left(t_i, b_i, g_i\right)$, $A_{(i,j)}^k \equiv \left\{B_x,\ 1 \le x \le Q_{(i,j)}^k\right\}$. As the CAFs are wide-sense increasing, the abscissas of the breakpoints are strictly increasing, and $b_{i+1} \geq b_i + g_i$. 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 three 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 [5], 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 underlying FIFO hypothesis: more specifically, for all flows (i,j) traversing node k as part of a FIFO aggregate, $\forall t \geq 0$ $D^k_{(i,j)}(t) = A^k_{(i,j)}(x(t))$, where $x(t) = \sup\{\tau \leq t : A^k(\tau) = D^k(t)\}$. 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 $\left[(D^k)^{-1}(b_0),(D^k)^{-1}(b_1)\right)$ 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 $R^{out}\cdot 1_{\{(i,j)\equiv f\}}$. Figure 19 reports a graphic representation of the FIFO multiplexing and de-multiplexing of two CAFs.

To the best of our knowledge, few other software tools have been developed for Network Calculus problems. The DISCO calculator, [18], implements some basic Network Calculus operations on curves, such as sum, minimum, convolution, deconvolution and sub-additive closure. However, it assumes *blind* multiplexing (instead of FIFO). Furthermore, it does not compute CDFs from CAFs: rather, it computes *output arrival curves* from (concave) input arrival curves. The COINC library [19] implements basic (min, +) algebra operations, hence – although not a tool itself, lacking network representation, it can be used to build a tool. The RTC [27] and CyNC [28] toolboxes allow one to compute CDFs from CAFs through (min,+) convolution. However, as far as we

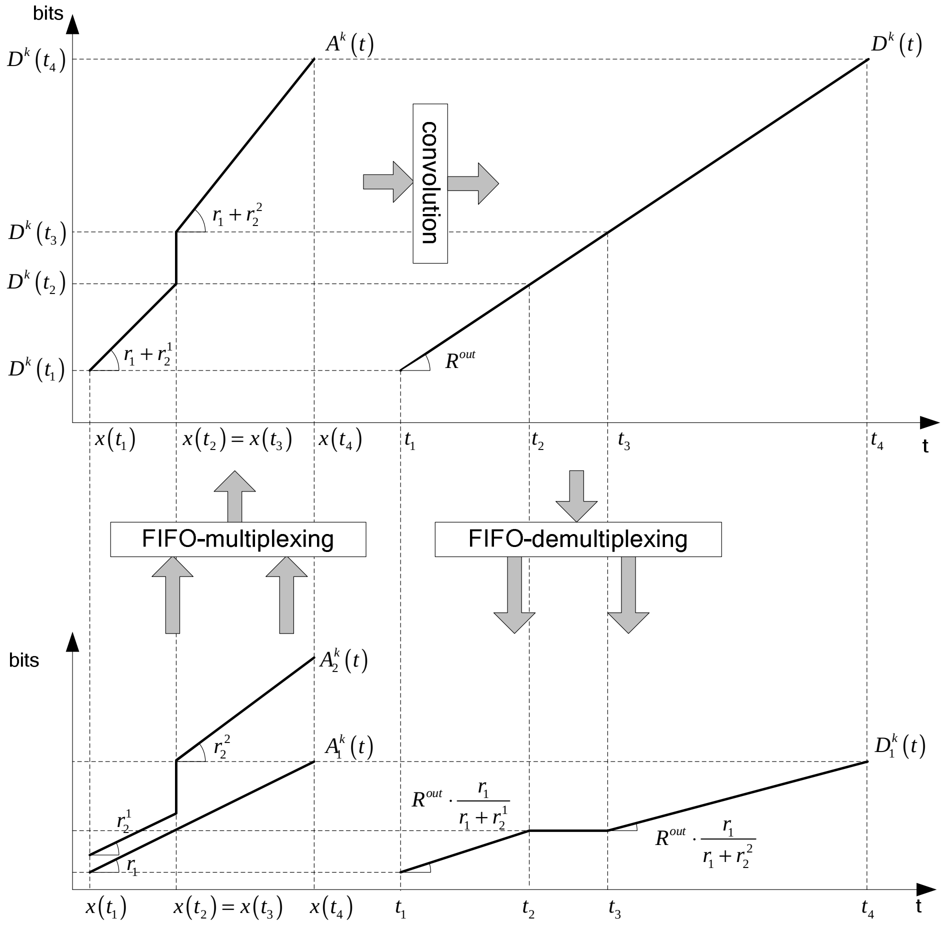


Figure 19 –CAF 1 is multiplexed with CAF 2 at a FIFO node and transformed in the related CDF

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.

6.2 Description of the Scenario

The hypotheses based on which we build the lower bound scenario are the same that were proved in [12] to actually represent *the* worst-case scenario for sink-tree tandems. While this does not imply that the same holds for generic tandems, it nonetheless provides a good motivation. The hypotheses are the following:

a) All nodes are lazy. This is not restrictive by Lemma 5.1.

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 $\sigma_{(1,N)}-1$.

c) Every cross flow (i,j) sends "as much traffic as possible", so as to delay the $\sigma_{(1,N)}$ th bit of the tagged flow.

We measure the delay experienced by the $\sigma_{(1,N)}$ th bit of the tagged flow under these hypotheses.

Let us take a closer look at hypothesis c) above. Call a^x, b^x the time instants when the *first* and the *last* bit of the tagged flow arrive at node x. For instance, it is $a^1=b^1=0$, while $a^x<b^x$ for $x>1$, since all nodes are lazy. Hypothesis c) implies that

$$A_{(i,j)}^{i}\left(b^{i}\right)-A_{(i,j)}^{i}\left(a^{i}\right)=\alpha_{(i,j)}\left(b^{i}-a^{i}\right) \tag{34}$$

for each flow (i,j). However, there are infinite CAFs that verify (34). For instance, one is the *greedy* CAF, $A_{(i,j)}^{i}(t)=\alpha_{(i,j)}\left(t-a^{i}\right)$, while another one is $A_{(i,j)}^{i}(t)=F_{(i,j)}^{i}(t)$, with:

$$F_{(i,j)}^{i}(t)=\begin{cases}\rho_{(i,j)}\cdot\left(t-a^{i}\right)^{+} & t<b^{i}\\ \rho_{(i,j)}\cdot\left(b^{i}-a^{i}\right)+\sigma_{(i,j)} & t=b^{i}\end{cases} \tag{35}$$

which we call *delayed greedy* CAF, in which the flow sends its burst $\sigma_{(i,j)}$ *just before* the $\sigma_{(1,N)}$ th bit of the tagged flow arrives at node i, as shown in Figure 20.

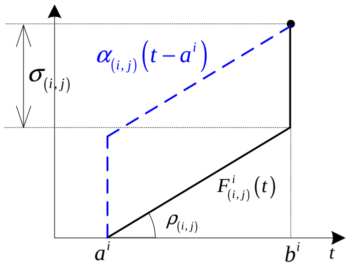


Figure 20 - Cumulative arrival functions for flow *(i,j)*

Under the hypotheses of the system model, if all the CAFs for the cross flows are either greedy or delayed greedy, both the *total* CAF and the CAF of the tagged flow at each node are piecewise linear. Furthermore, v is the time instant when the $\sigma_{(1,N)}$ th bit of the tagged flow leaves node N.

It turns out that, depending on the values associated to the nodes and flows parameters, using either the greedy or the delayed greedy CAF for the cross flows actually leads to different delays, and it is not always possible to establish which is the largest beforehand. For instance, in sink-tree networks the WCD is achieved with delayed-greedy arrivals at all flows, although in some cases swapping a delayed-greedy CAF for a greedy one at some cross flow does not change the result [12]. Testing both greedy and delayed greedy CAFs for each cross-flow entails testing up to 2^M different scenarios, which clearly represents a problem as the number of flows increases. Devising topological properties that allow the number of scenarios to be reduced is part of the ongoing work. In the next subsection, we assess the tightness of the upper bounds for the tandems analyzed in Sections 4.1 and 5.

6.3 Assessing the Tightness of the Upper Bounds

In order to show that, on one hand, FE is effective in complementing LUDB, and, on the other hand, that the heuristics behind the computation of the lower bound are effective, we first compute a lower bound for the tandems shown in the two examples of Section 5. The results are as follows:

- Example 5.3: $v=\wedge\left(V,\bar{V}\right)$ in regions I, II, V. In the above cases, flow $(1,1)$ can be assumed to be either greedy or delayed-greedy indifferently, since $a^1=b^1=0$. In regions III and IV it is

$$v=\sum_{i=1}^{2}\theta^{i}+\frac{\sigma_{(1,1)}}{R^{1}}+\frac{\sigma_{(1,2)}}{R^{2}}<\wedge\left(V,\bar{V}\right)$$

- Example 5.4: when (32) holds, it is always $v=\bar{V}$. When (33) holds, instead, it is $v=\bar{V}$ only when $R^2\le R^3$, otherwise it is:

$$v=\sum_{i=1}^{3}\theta^{i}+\frac{\sigma_{(2,3)}}{R^{3}}+\frac{\sigma_{(1,3)}}{R^{3}\cdot\frac{R^{1}}{R^{1}+\rho_{(2,3)}}}+\frac{\sigma_{(1,2)}}{R^{2}\cdot\frac{R^{1}}{R^{1}+\rho_{(2,3)}}}<\wedge\left(V,\bar{V}\right)$$

Whenever $v=\bar{V}$, the WCD is obtained when flow (2,3) is delayed greedy (flow (1,2) can be considered either way, as specified before). However, when (33) holds, a greedy CAF for flow (2,3) yields the same result.

We then compute the ROB for the nested tandems used as case studies in Section 4.1, i.e. those having balanced k-ary nesting trees with a level of nesting equal to l. For each value of l,k, we instantiate 50 tandems with random nodes and flows parameters (as described in Section 4.1), we compute the LUDB (not trying FE), and report the average and maximum ROB. The results are shown in Table 3 The lower bound, and, accordingly, the ROB, is *exact* for the first three set of topologies. When the number of scenarios gets too large (as in the last two rows of the table, where it is $64\cdot 10^9$ and $128\cdot 10^{15}$ respectively), DEBORAH can be configured to select a fixed number of scenarios (which it picks uniformly at random among all scenarios) so as to keep the computation time reasonable. When this happens, the maximum ROB is underestimated, and the average ROB is not necessarily reliable. The results in the last two rows of Table 3. were obtained testing $64\cdot 10^6$ combinations.

l	***k***	**max ROB**	**avg ROB**
3	2	0.474	0.317
4	2	0.341	0.329
2	3	0.458	0.245
3	3	≥ 0.261	0.257 (*)
5	2	≥ 0.402	0.383 (*)

Table 3 – ROB for some nested tandems having balanced nesting trees

In all cases, the ROB stays below 0.5, which means that the LUDB is less than twice the WCD.

We report three more case studies. The first two are related to nested tandems, one having one-hop persistent cross-traffic, and the other in a *source-tree* topology. The last one deals with the most unfavorable non-nested tandem.

***Case study* 1 – one-hop persistent cross traffic**

We analyze a tandem of N nodes, traversed by the tagged flow $(1,N)$ and by cross-flows (i,i), $1 \le i < N$, shown in Figure 21. We assume that all flows have the same leaky-bucket arrival curve, with $\sigma = 5$ and $\rho = 4$. All nodes have the same rate-latency service curve, with $\theta = 1$ and $R = 2\rho/U$, U ranging from 20% to 100%. The LUDB expression for that tandem is available in a closed form, and it is equal to (see Theorem 2 in [11]):

$$V = N \cdot \left[\theta + \frac{U \cdot \sigma}{\rho} \cdot \left(\frac{1}{2} + \frac{1}{2-U} \right) \right]$$

Figure 22 shows the ROB as a function of the number of nodes and for various values of U. As the figure shows, the ROB increases with both N and U. However, it tends to reach a limit value as N grows higher. While the exact quota of the ROB depends on the actual parameter values, the same behavior is always observed.

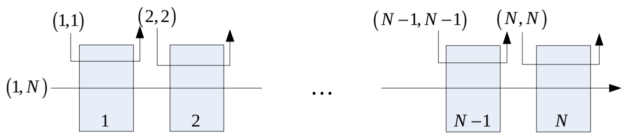


Figure 21 – A case-study nested tandem with one-hop persistent cross-traffic

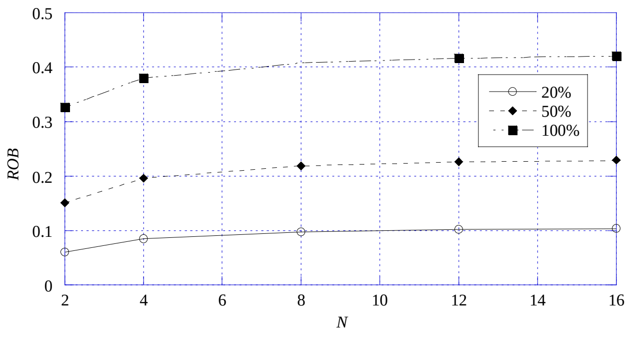


Figure 22 – ROB in the nested case-study tandem

Note that we can apply FE to the above tandem when $U < 100\%$. For instance, when $N = 8$, $U = 20\%$ all the cross flows can be extended to the last node, thus yielding a sink-tree tandem with a tagged flow $(1,N) \equiv (2\sigma, 2\rho)$ and cross-flows $(i,N) \equiv (\sigma,\rho)$, $2 \le i \le 8$, for which the LUDB can be computed in a closed form applying the formula in [12]. As shown in Table 4, this improves the ROB of about 40%.

tandem	LUDB	Lower Bound	ROB
original	10.111	9.125	9.75%
with FE	9.673		5.66%

Table 4 – ROB for the one-hop persistent nested tandem with N=8 and U=20%

€

***Case study* 2 – source tree tandem**

The case-study N-node tandem, shown in Figure 23, is traversed by the tagged flow $(1,N)$ and by cross-flows $(1,i)$, $1 \le i < N$. We call such a fully-nested tandem a *source-tree* tandem, for symmetry with the sink-tree case. All flows have the same leaky-bucket arrival curve, with $\sigma = 5$ and $\rho = 4$. Nodes have a rate-latency service curve, with $\theta^i = 1$ and $R^i = (N+1-i) \cdot \rho/U$, U ranging from 20% to 100%. The LUDB for that tandem can be derived in a closed form by considering that all fully nested tandems are *tree equivalent* (see [14]) and using the known formulas for sink-trees [12]. In the above settings, we have:

$$V = N \cdot \theta + \frac{U \cdot \sigma}{\rho} \cdot H_N ,$$

where H_N is the *N*-th harmonic number. Figure 26 shows the ROB as a function of the number of nodes and for various values of U. As the figure shows, the ROB increases with U. However, on one hand the ROB values are smaller than in the former case. On the other hand, they peak at $N = 4$ and then decrease afterwards.

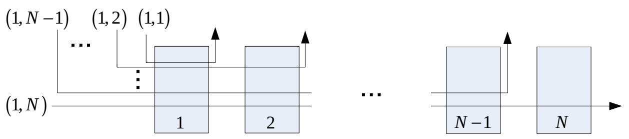


Figure 23 – A case-study source-tree nested tandem

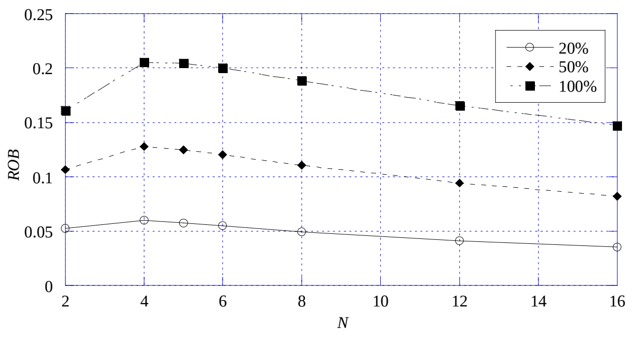


Figure 24 – ROB in the source-tree case-study tandem

Note that, in this particular case, FE does not yield any improvement in the delay bounds. For instance, when $U \le 1/N$ all the cross flows can be extended to the last node, thus yielding a tandem with only a tagged flow $(1,N) \equiv (N \cdot \sigma, \underline{N} \cdot \rho)$ and no cross-flows. For this, the LUDB is trivially equal to $\overline{V} = N \cdot (\theta + U \cdot \sigma/\rho)$, and it is $\overline{V} \ge V$ since $H_N \le N$.

€

***Case study* 3 – non-nested tandem**

We now analyze a very unfavorable scenario, shown in Figure 25, where an even number of nodes N are traversed by a tagged flow $(1,N)$, by all flows $(i,i+1)$, $1 \le i < N$, and by two flows $(1,1)$ and (N,N) for symmetry. Computing the LUDB in the latter requires the maximum possible number of cuts and partial delay bound computations, i.e. $N/2$. Thus, it is very likely that the LUDB is overrated, all the more as N grows higher. We assume that all flows have the same leaky-bucket arrival curve, with $\sigma = 5$ and $\rho = 4$. All nodes have the same rate-latency service curve, with $\theta = 1$ and $R = 3\rho/U$, U ranging from 20% to 100%. The recursive LUDB expression for that tandem is the following (see [14] for the computations):

$$V = \sum_{i=1}^{N/2} \left(2\theta + \frac{\sigma + \sigma_i^f}{R} + \frac{\sigma + \sigma_i^t}{R - \rho} \right), \qquad (36)$$

With:

$$\sigma_1^t = \sigma_1^f = \sigma\,,\ \sigma_{i+1}^f = \sigma + \rho \cdot \left[\theta + \frac{\sigma + \sigma_i^t + 2\rho \cdot \left(\theta + \sigma_i^f / R \right)}{R} \right]$$

$$\sigma_{i+1}^t = \sigma_i^t + \rho \cdot \left[2\theta + \frac{\sigma + \sigma_i^f}{R} + \frac{\sigma}{R - \rho} \right]$$

Figure 26 shows the ROB as a function of the number of nodes and for various values of U. As the figure shows, the gap grows with N and with U. This confirms what already said in Section 5 on the non tightness of the LUDB in non-nested tandems.

€

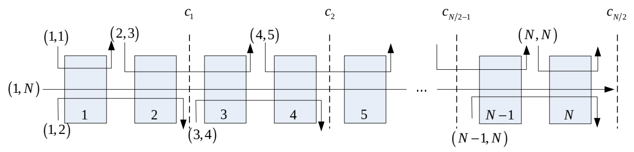


Figure 25 – A case-study non-nested tandem

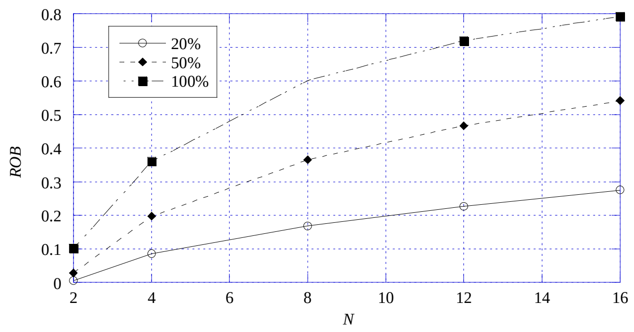


Figure 26 – ROB in the non-nested case-study tandem

Once more, we can apply FE to the above tandem when $U < 100\%$. For instance, when $N = 8$, $U = 20\%$ all the cross flows $(i, i+1)$ can be extended to the last node. This yields again a sink-tree tandem, with a tagged flow $(1,N) \equiv (3\sigma, 3\rho)$ and cross-flows $(i,N) \equiv (\sigma,\rho)$, $2 \le i \le 8$. As shown in Table 5, this improves the ROB of about 85%.

tandem	LUDB	Lower Bound	ROB
original	10.666	8.872	16.78%
with FE	9.106		2.57%

Table 5 – ROB for the non-nested tandem with N=8 and U=20%

€

Within the limits of the analyzed case studies, some conclusive remarks can be attempted. For nested tandems, the LUDB appears to be of the same order of magnitude as the WCD. In fact, as N grows larger, the ROB does not appear to approach the unity. However, as the provisioning gets tighter, the uncertainty about the tightness increases as well. In non-nested tandems, the ROB appears to grow faster, confirming the intuition that end-to-end analysis is necessary to achieve reliable bounds. Although it would be tempting to try to infer general relationships linking the tandem *topology* (i.e., the shape of the nesting tree) to the tightness of the delay bounds in nested tandems, we remark that the tightness also depends on the *flows* and *nodes'* rates (see, for instance, Example 5.3 and Example 5.4), so that different ROBs can be obtained for the same topology just by varying the rates.

7. Using DEBORAH

In this section we briefly 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.

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. The heuristic approximation 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.

When LUDB computation is invoked, the tool first checks whethter the tandem is non-nested. In that case, it 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.

8. CONCLUSIONS

Following our previous work [14], this paper has addressed the problem of how to practically compute the least upper delay bound (LUDB) for a flow traversing a FIFO-multiplexing tandem, and how to assess whether the latter is equal to (or, as a subordinate, close to) the actual worst-case delay. As far as the first problem is concerned, we have developed a tool which allows both exact and heuristically approximated LUDB computation. The exact algorithm solves a possibly large number of simplexes, doing its best to avoid infeasible ones, while the heuristic algorithm limits the number of simplexes to be solved, trying to pick up those which are more likely to yield the actual LUDB. The latter has been shown to provide very good approximations of the LUDB at a small computational cost. As far as tightness is concerned, we have shown that the current Network Calculus theorems related to FIFO multiplexing are not sufficient for computing the worst-case delay in tandem networks. In fact, the LUDB itself can sometimes be improved upon, even in very simple cases. We have shown this introducing a method – called Flow Extension – that allows one to compute delay bounds by exploiting topological properties of tandems. We have then addressed the question of how close the upper bounds are to the (still unknown) worst-case delay. We have devised an algorithm to compute *lower* bounds on the worst-case delay. The analysis reported in the paper show that the upper and lower bounds are of the same order in nested tandems, while they tend to diverge in non-nested tandems, where end-to-end analysis is not possible. This further confirms the common belief that end-to-end analysis is fundamental to achieve a reasonably tight worst-case delay assessment.

9. REFERENCES

[1] L. Bisti, L. Lenzini, E. Mingozzi, G. Stea, “Estimating the Worst-case Delay in FIFO Tandems Using Network Calculus”, VALUETOOLS 2008, Athens, Greece, 21-23 October 2008

[2] L. Bisti, L. Lenzini, E. Mingozzi, G. Stea, “DEBORAH: A Tool for Worst-Case Analysis of FIFO Tandems”, ISoLA 2010, Crete, October 2010

[3] R. Braden, D. Clark and S. Shenker, “Integrated Services in the Internet Architecture: an Overview”, IETF RFC 1633, June 1994.

[4] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, “An Architecture for Differentiated Services,” IETF RFC 2475, 1998.

[5] J.-Y. Le Boudec, P. Thiran, Network Calculus, Springer-Verlag LNCS vol. 2050, 2001.

[6] E. Rosen, A. Viswanathan, R. Callon, “Multiprotocol Label Switching Architecture”, IETF RFC 3031, January 2001

[7] R.L. Cruz. “A calculus for network delay, part i: Network elements in isolation”. IEEE Transactions on Information Theory, Vol. 37, No. 1, March 1991, pp. 114-131.

[8] R.L. Cruz. “A calculus for network delay, part ii: Network analysis”. IEEE Transactions on Information Theory, Vol. 37, No. 1, March 1991, pp. 132–141.

[9] R. Agrawal, R. L. Cruz, C. Okino, and R. Rajan, “Performance Bounds for Flow Control Protocols,” IEEE/ACM Transactions on Networking, Vol. 7, No. 3, June 1999, pp. 310-323.

[10] C. S. Chang, Performance Guarantees in Communication Networks, Springer-Verlag, New York, 2000.

[11] L. Lenzini, E. Mingozzi, G. Stea, “Delay Bounds for FIFO Aggregates: a Case Study”, Elsevier Computer Communications Vol. 28 Issue 3, February 2005 pp. 287–299.

[12] L. Lenzini, L. Martorini, E. Mingozzi, G. Stea, “Tight End-to-end Per-flow Delay Bounds in FIFO Multiplexing Sink-tree Networks", Performance Evaluation, Vol. 63, October 2006, pp. 956-987.

[13] L. Lenzini, E. Mingozzi, G. Stea, "End-to-end Delay Bounds in FIFO-multiplexing Tandems", VALUETOOLS'07, Nantes (FR), October 23-25, 2007

[14] L. Lenzini, E. Mingozzi, G. Stea, "A Methodology for Computing End-to-end Delay Bounds in FIFO-multiplexing Tandems", to appear on Performance Evaluation, 2008 (already available at http://www.sciencedirect.com)

[15] M. Fidler, V. Sander, “A Parameter Based Admission Control for Differentiated Services Networks”, Elsevier Computer Networks, Vol. 44, No 1, January 2004, pp. 463-479.

[16] R. L. Cruz. “Sced+: Efficient management of quality of service guarantees”. Proc. of IEEE Infocom’98, San Francisco (USA), 29 March-April 1998, pp. 625-634.

[17] J. C. R. Bennett, K. Benson, A. Charny, W. F. Courtney, and J.-Y. Le Boudec, “Delay Jitter Bounds and Packet Scale Rate Guarantee for Expedited Forwarding,” IEEE/ACM Trans. on Networking, Vol. 10, No. 4, August 2002, pp. 529-540.

[18] J. B. Schmitt, F. A. Zdarsky, “The DISCO Network Calculator - A Toolbox for Worst Case Analysis” Proc. of VALUETOOLS '06, Pisa, Italy. ACM, October 2006.

[19] Anne Bouillard, Éric Thierry, “An Algorithmic Toolbox for Network Calculus”, INRIA research report 6094, 2007, to appear in Journal of Discrete Event Dynamic Systems.

[20] Website of the Computer Networking Group at the University of Pisa, http://info.iet.unipi.it/~cng/, continuously updated

[21] G. Urvoy-Keller, G. Hèbuterne, Y. Dallery, “Traffic Engineering in a Multipoint-to-point network.”, *IEEE Journal on Selected Areas in Communications, Special Issue on Recent Advances in Network Management,* Vol. 20, No. 4, May 2002, pp. 834-849

[22] J. Schmitt and U. Roedig, “Sensor network calculus - a framework for worst case analysis,” in *Proc. Distributed Computing on Sensor Systems (DCOSS)*, pp. 141–154, June 2005.

[23] A. Koubaa, M. Alves, and E. Tovar, “Modeling and worst-case dimensioning of cluster-tree wireless sensor networks,” in *Proc. IEEE RTSS*, pp. 412–421, 2006.

[24] T. Skeie, S. Johannessen, and O. Holmeide, “Timeliness of real-time IP communication in switched industrial Ethernet networks,” *IEEE Transactions on Industrial Informatics*, vol. 2, pp. 25–39, Feb. 2006.

[25] S. Chakraborty, S. Kuenzli, L. Thiele, A. Herkersdorf, and P. Sagmeister, “Performance evaluation of network processor architectures:

Combining simulation with analytical estimation," *Computer Networks*, vol. 42, no. 5, pp. 641–665, 2003.

[26] R. Pastor-Satorras, A. Vespignani, "Evolution and Structure of the Internet: A Statistical Physics Approach", Cambridge University Press, 2004

[27] E. Wandeler, L. Thiele, "Real-Time Calculus (RTC) Toolbox", available online at `http://www.mpa.ethz.ch/Rtctoolbox`, 2006

[28] H. Schioler, H.P. Schwefel, M.B. Hansen "CyNC – a MATLAB/Simulink Toolbox for Network Calculus", Proc. VALUETOOLS'07, Nantes (FR), October 2007

10. APPENDIX

We report here the expressions for the LUDB in the non-nested tandem dealt with in Example 5.4 (see [14] for the computations). Delay bound V^a is obtained by using $\{3,4\}$ as a set of cuts, and its expression is the following.

If $R^1 + \rho_{(2,3)} < R^2$,

$$\begin{aligned} V_1^a = {} & \theta^1 \cdot \left[1 + \frac{\rho_{(1,3)}}{R^3} \cdot \left(1 + \frac{\rho_{(2,3)}}{R^2}\right)\right] + \theta^2 \cdot \left(1 + \frac{\rho_{(2,3)} + \rho_{(1,3)}}{R^3}\right) + \theta^3 \\ & + \frac{\sigma_{(1,2)}}{R^3}\left(\frac{\rho_{(2,3)}}{R^2} + \frac{R^3 + \rho_{(1,3)}}{R^1}\right) \\ & + \frac{\sigma_{(1,3)}}{R^3} \cdot \left(1 + \frac{\rho_{(2,3)}}{R^2}\right) + \frac{\sigma_{(1,3)}}{R^1} + \frac{\sigma_{(2,3)}}{R^2} \cdot \left(1 + \frac{R^2 + \rho_{(1,3)}}{R^3}\right) \end{aligned} \tag{37}$$

Otherwise,

$$\begin{aligned} V_2^a = {} & \theta^1 \cdot \left[1 + \frac{\rho_{(1,3)}}{R^3} \cdot \left(1 + \frac{\rho_{(2,3)}}{R^2}\right)\right] + \theta^2 \cdot \left(1 + \frac{\rho_{(2,3)} + \rho_{(1,3)}}{R^3}\right) + \theta^3 \\ & + \frac{\sigma_{(1,2)}}{R^2} \cdot \left[\frac{\rho_{(2,3)}}{R^3} + \left(1 + \frac{\rho_{(1,3)}}{R^3}\right) \cdot \left(1 + \frac{\rho_{(2,3)}}{R^1}\right)\right] \\ & + \frac{\sigma_{(1,3)}}{R^2} \cdot \left(1 + \frac{\rho_{(2,3)}}{R^2} + \frac{R^2 + \rho_{(2,3)}}{R^3}\right) + \frac{\sigma_{(2,3)}}{R^2} \cdot \left(1 + \frac{R^2 + \rho_{(1,3)}}{R^3}\right) \end{aligned} \tag{38}$$

Delay bound V^b is instead obtained by using $\{2,4\}$ as a set of cuts, and its expression is the following.

If $R^3 + \rho_{(1,2)} < R^2$,

$$\begin{aligned} V_1^b = {} & \theta^1 \cdot \left(1 + \frac{\rho_{(1,2)}}{R^2} + \frac{\rho_{(1,3)}}{R^3}\right) + \theta^2 + \theta^3 \\ & + \frac{\sigma_{(1,2)}}{R^1}\left(1 + \frac{\rho_{(1,3)}}{R^3}\right) + \frac{\sigma_{(1,2)}}{R^2} + \frac{\sigma_{(1,3)}}{R^1}\left(1 + \frac{\rho_{(1,2)}}{R^2}\right) + \frac{\sigma_{(1,3)}}{R^3} + \frac{\sigma_{(2,3)}}{R^3} \end{aligned} \tag{39}$$

Otherwise,

$$\begin{aligned} V_2^b = {} & \theta^1 \cdot \left[1 + \frac{\rho_{(1,2)}}{R^2} + \frac{\rho_{(1,3)}}{R^2} \cdot \left(1 + \frac{\rho_{(2,3)}}{R^3}\right)\right] + \theta^2 + \theta^3 \\ & + \frac{\sigma_{(1,2)}}{R^1} \cdot \left[1 + \frac{\rho_{(1,3)}}{R^2} \cdot \left(1 + \frac{\rho_{(2,3)}}{R^3}\right)\right] + \frac{\sigma_{(1,2)}}{R^2} + \frac{\sigma_{(1,3)}}{R^1} \cdot \left(1 + \frac{\rho_{(1,2)}}{R^2}\right) \\ & + \frac{\sigma_{(1,3)}}{R^2} \cdot \left(1 + \frac{\rho_{(2,3)}}{R^3}\right) + \frac{\sigma_{(2,3)}}{R^2} \cdot \left(1 + \frac{\rho_{(2,3)}}{R^3}\right) \end{aligned} \tag{40}$$