Delay Bounds under Arbitrary Multiplexing*

Jens B. Schmitt[1], Frank A. Zdarsky[1], Markus Fidler[2]

[1]disco | Distributed Computer Systems Lab, University of Kaiserslautern, Germany
[2]Multimedia Communications Lab, Darmstadt University of Technology, Germany

Technical Report No. 360/07

July 2007

Abstract. Network calculus has proven as a valuable and versatile methodology for worst-case analysis of communication networks. One issue in which it is still lacking is the treatment of aggregate multiplexing, in particular if the FIFO property cannot be assumed when flows are merged. In this report, we address the problem of bounding the delay of individual traffic flows in feed-forward networks under arbitrary multiplexing. Somewhat surprisingly, we find that direct application of network calculus results in loose bounds even in seemingly simple scenarios. The reasons for this "failure" of network calculus are discussed in detail and a method to arrive at tight delay bounds for arbitrary (aggregate) multiplexing is presented. This method is based on the solution of an optimization problem. For the special case of sink-tree networks this optimization problem is solved explicitly, thus arriving at a closed-form expression for the delay bound. Numerical experiments illustrate that in sink-tree networks the improvement over bounds based on direct application of network calculus can be considerable.

Keywords: Performance Evaluation, Performance Bounds, Network Calculus, Feed-forward Networks, Arbitrary Multiplexing.

* The research in this report is supported in part by an Emmy Noether grant of the German Research Foundation (DFG).

1 Introduction

Network calculus is a min-plus system theory for deterministic queuing systems which builds on the calculus for network delay in [1], [2]. The important concept of *service curve* was introduced in [3,4,5,6,7]. The service curve based approach facilitates the efficient analysis of tandem queues where a linear scaling of performance bounds in the number of traversed queues is achieved as elaborated in [8] and also referred to as pay bursts only once phenomenon in [9]. A detailed treatment of min-plus algebra and of network calculus can be found in [10] and [9], [11], respectively.

Network calculus has found numerous applications, most prominently in the Internet's Quality of Service (QoS) proposals IntServ and DiffServ, but also in other scenarios as, for example, wireless sensor networks [12,13], switched Ethernets [14], Systems-on-Chip (SoC) [15], or even to speed-up simulations [16]. Hence, besides queueing theory it has established as a valuable methodology.

However, as a relatively young theory, compared to e.g. traditional queueing theory, there is also a number of challenges network calculus still has to master. To name a few: recently there has been much interest and progress with respect to stochastic extensions (see [8], [17], [18] for recent advances); tool support for network calculus has been addressed by [19], [20] and brings about new interesting perspectives. A very tough challenge is also found in the treatment of non-tandem topologies with aggregate multiplexing of multiple flows. While this has been addressed from the beginning [2], there are still many open issues. For aggregate multiplexing in general network topologies there is a very fundamental issue about the circumstances under which a finite delay bound exists at all [21], [22]. In [23] a sufficient condition for stability in general network topologies and an explicit delay bound are given. Extensions of this approach are provided by [24] and [25]. Yet, for larger networks this puts a heavy constraint on the utilization of the network since the maximum allowable utilization is inversely proportional to the network diameter. The problems in the analysis of general topologies arise due to cyclic dependencies between flows and the consequent difficulties in bounding their network-internal burstiness. A special class of topologies which avoids those problems are feed-forward networks, which are known to be stable for all utilizations$\leq$1 [2]. In this report, we focus on this class of networks. While many networks are obviously not feed-forward, many important instances like switched networks, wireless sensor networks, or MPLS networks with multipoint-to-point label switched paths are, or can be made feed-forward by using, e.g., the turn-prohibition algorithm [26].

In feed-forward networks, there has been some work on aggregate multiplexing recently: [27] treats the case of feed-forward networks under FIFO multiplexing for token-bucket constrained flows and rate-latency servers. The left-over service curve for a flow of interest is derived. It is again of the rate-latency type with minimally possible latency. [28] shows that this does not result in a tight delay bound, and derives tight delay bounds under knowledge about the arrival curve of the flow of interest for the special case of sink-trees and, again, under token bucket constrained flows and rate-latency servers. Another work [29] also investigates sink-tree networks, but now under dual token-bucket constrained flows and constant rate servers, for which delay bounds are derived by summing per-node bounds, expectedly not arriving at tight bounds but reported as being at least close under practical conditions.

Besides being very specific with respect to traffic and server models, all of the above work assumes FIFO aggregate multiplexing. However in practice, as argued in [30], many devices cannot be accurately described by FIFO because packets arriving at the output queue from different input ports may experience different delays when traversing a node. This is due to the fact that many networking devices like routers are implemented using input-output buffered crossbars and/or multistage interconnections between input and output ports. Hence, packet reordering on the aggregate level is a frequent event (not so on the flow level) and should not be neglected in modelling. Therefore, in this work we drop the FIFO multiplexing assumption and make essentially no assumptions on the way aggregates are multiplexed at servers, i.e. we assume arbitrary multiplexing a.k.a general or blind multiplexing [1], [9]. On the level of a single flow, however, we still assume FIFO. This assumption is sometimes called FIFO-per-microflow [31] or locally FCFS multiplexing [1].

There is actually little work on arbitrary multiplexing: Some results are reported in [9] (see Section 2), and there is some work on the burstiness increase due to arbitrary multiplexing at a single node [32]. In previous work [19] related to network calculus tool support, we have proposed and implemented a number of network calculus analysis methods for arbitrary multiplexing feed-forward networks. Some of these are presented in Section 3, but as will be demonstrated, they were not the “final word on this topic”. While not addressing aggregate multiplexing, we also mention [31] here, because it demonstrates that releasing FIFO assumptions can lead to interesting and somewhat unexpected phenomena, similar to what we will be dealing with later on.

The goal of our work is to calculate tight delay bounds in feed-forward networks of arbitrary multiplexers. With respect to traffic and server models we address a more general case than previous work on FIFO multiplexing, in particular we assume piecewise linear concave arrival curves and convex service curves, which encompass the majority of practical traffic and server models.

In essence, the main contributions of this report are

– exposition of a fundamental problem of network calculus to achieve tight delay bounds in any non-FIFO aggregate multiplexing (→Section 3);
– a novel method to achieve tight delay bounds in feed-forward networks under arbitrary multiplexing (→Section 4);
– closed-form expressions for tight delay bounds in sink-tree networks (→Section 5).

2 Network Calculus Basics

As network calculus is built around the notion of cumulative functions for input and output flows of data, the set of real-valued, non-negative, and wide-sense increasing functions passing through the origin plays a major role:

$$\mathcal{F} = \{f : \mathbb{R}^+ \to \mathbb{R}^+, \forall t \geq s : f(t) \geq f(s), f(0) = 0\}$$

In particular, the input function $F(t)$ and the output function $F'(t)$, which cumulatively count the number of bits that are input to, respectively output from, a system $\mathcal{S}$, are in $\mathcal{F}$. Throughout the report, we assume in- and output functions to be continuous in time and space. Note that this is not a general limitation as there exist transformations between discrete and continuous time models [9].

Definition 1. *(Min-plus Convolution and Deconvolution) The min-plus convolution respectively deconvolution of two functions f and g are defined to be*

$$(f \otimes g)(t) = \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\}$$

$$(f \oslash g)(t) = \sup_{u \geq 0} \{f(t+u) - g(u)\}$$

It can be shown that the triple $(\mathcal{F}, \wedge, \otimes)$, where $\wedge$ denotes the minimum operator (which ought to be taken pointwise for functions), constitutes a dioid [9]. Also, the min-plus convolution is a linear operator on the dioid $(\mathbb{R} \cup \{+\infty\}, \wedge, +)$, whereas the min-plus deconvolution is not. These algebraic characteristics result in a number of rules that apply to those operators, many of which can be found in [9], [11]. Let us now turn to the performance characteristics of flows which can be bounded by network calculus means:

Definition 2. *(Backlog and Delay) Assume a flow with input function F that traverses a system $\mathcal{S}$ resulting in the output function F'. The* backlog *of the flow at time t is defined as*

$$x(t) = F(t) - F'(t)$$

Assuming FIFO delivery, the virtual delay *for a bit input at time t is defined as*

$$d(t) = \inf \{\tau \geq 0 : F(t) \leq F'(t+\tau)\}$$

Next, the arrival and departure processes specified by input and output functions are bounded based on the central network calculus concepts of arrival and service curves:

Definition 3. *(Arrival Curve) Given a flow with input function F a function $\alpha \in \mathcal{F}$ is an arrival curve for F iff*

$$\forall t, s \geq 0, s \leq t : F(t) - F(t-s) \leq \alpha(s) \Leftrightarrow F \leq F \otimes \alpha$$

A typical example of an arrival curve is given by an affine arrival curve $\gamma_{r,b}(t) = b + rt$, $t > 0$ and $\gamma_{r,b}(t) = 0$, $t \leq 0$ which corresponds to token-bucket traffic regulation.

Definition 4. *(Service Curve) If the service provided by a system $\mathcal{S}$ for a given input function F results in an output function F' we say that $\mathcal{S}$ offers a service curve β iff*

$$F' \geq F \otimes \beta$$

A typical example of a service curve is given by a so-called rate-latency function $\beta_{R,T}(t) = R[t-T]^+$, where the $[.]^+$operator is defined as $[x]^+ = x \vee 0$, and $\vee$ denotes the maximum operator. A number of systems fulfill, however, a stricter definition of service curve [9], which is particularly useful as it permits certain derivations that are not feasible under the more general minimum service curve model.

Definition 5. *(Strict Service Curve) Let $\beta \in \mathcal{F}$. System $\mathcal{S}$ offers a* strict *service curve β to a flow if, during any backlogged period of duration u the output of the flow is at least equal to $\beta(u)$.*

Note that any strict service curve is also a service curve, but not the other way around. Many schedulers offer strict service curves, for example most of the generalized processor sharing-emulating schedulers offer a strict service curve of the rate-latency type. Strict service curves will play a crucial role in this report, since they, in contrast to service curves, allow to bound the maximum backlogged period of a system. More specifically, that bound $\bar{d}$ is given as the non-zero intersection point between arrival and service curve, i.e. $\alpha(\bar{d}) = \beta(\bar{d})$.

Using those concepts it is possible to derive *tight* performance bounds on backlog, (virtual) delay and output:

Theorem 1. *(Performance Bounds) Consider a system $\mathcal{S}$ that offers a service curve β. Assume a flow F traversing the system has an arrival curve α. Then we obtain the following performance bounds:*
Backlog: $\forall t : x(t) \leq (\alpha \oslash \beta)(0) =: v(\alpha, \beta)$
Delay: $\forall t : d(t) \leq \inf\{t \geq 0 : (\alpha \oslash \beta)(-t) \leq 0\} =: h(\alpha, \beta)$
Output (arrival curve α' for F'): $\alpha' = \alpha \oslash \beta$

Note that, if FIFO cannot be assumed, the bound on the maximum backlogged period under the assumption of a strict service curve can be used as an alternative delay bound instead of the horizontal deviation $h(\alpha, \beta)$.

One of the strongest results of network calculus (albeit being a simple consequence of the associativity of $\otimes$) is the concatenation theorem that enables us to investigate tandems of systems as if they were single systems:

Theorem 2. *(Concatenation Theorem for Tandem Systems) Consider a flow that traverses a tandem of systems $\mathcal{S}_1$ and $\mathcal{S}_2$. Assume that $\mathcal{S}_i$ offers a service curve β_i, $i = 1, 2$ to the flow. Then the concatenation of the two systems offers a service curve $\beta_1 \otimes \beta_2$ to the flow.*

Using the concatenation theorem, it is ensured that an end-to-end analysis of a tandem of servers still achieves tight performance bounds, which in general is not the case for an iterative per-node application of Theorem 1.

So far we have only covered the single flow case, the next result factors in the existence of other interfering flows. In particular, it states the minimum service curve available to a flow at a single node under cross-traffic from other flows at that node.

Theorem 3. *(Left-over Service Curve under Arbitrary Multiplexing) Consider a node multiplexing two flows 1 and 2 in arbitrary order. Assume that the node guarantees a* strict *minimum service curve β to the aggregate of the two flows. Assume that flow 2 has α_2 as an arrival curve. Then*

$$\beta^1 = [\beta - \alpha_2]^+$$

is a service curve for flow 1 if $\beta^1 \in \mathcal{F}$, often also called the left-over *service curve for the* flow of interest. *Note that we require the service curve to be strict. In [9], an example is given showing that the theorem otherwise would not hold.*

3 Network Calculus in Feed-Forward Networks

In this section, several methods of computing delay bounds in feed-forward networks of arbitrary multiplexers are presented. All of them are based on the direct application of well-known network calculus results.

3.1 Network Calculus Based Bounding Methods

The methods we present in the following compute delay bounds for a certain flow of interest. In order to do so all of them require to compute the network-internal flow constraints of each flow that is interfering with the flow of interest. This can be easily done in a feed-forward network by the application of the output bound from Theorem 1 and noting that the multiplexing of flows is performed by their addition.

For ease of exposition we now present the different alternatives in probably the most simple conceivable example scenario as illustrated in Figure 1. Despite its simplicity this scenario will already exhibit that any of the presented network calculus based methods runs into problems with respect to achieving tight delay bounds.

Total Flow Analysis (TFA) The first bounding method is probably the most direct application of basic network calculus results and is already mentioned in [2] to show that feed-forward networks are stable. The idea of this method is to compute per-node delay bounds for the total traffic offered to the respective node, which is why it has been called total flow analysis (TFA), and then sum those up for the end-to-end delay bound of the flow of interest. Under the assumption of arbitrary multiplexing this means the per-node delay bounds have to be computed as the maximum busy periods at the nodes because the total flow must be considered to be non-FIFO.

For the example scenario in Figure 1, the TFA delay bound for flow 1 can be computed as follows

$$d^{TFA} = d_1 + d_2 \quad \text{with}$$
$$(\alpha_1 + \alpha_2)(d_1) = \beta_1(d_1),\ ((\alpha_1 + \alpha_2) \oslash \beta_1)(d_2) = \beta_2(d_2)$$

Separated Flow Analysis (SFA) An obvious weakness of the TFA is that it makes no use of the concatenation theorem, which is known to provide a clear advantage over additive per-node bounds. To this end, instead of treating the total flow, the next method first separates the service provided to the flow of interest at each node, which is why it is called separated flow analysis (SFA), before applying the concatenation theorem to the tandem of left-over service curves. The separation is based on Theorem 3 for the left-over service at a single node under arbitrary multiplexing. Note that in contrast to the TFA, the horizontal deviation can be used for the calculation of the end-to-end delay bound because each flow is assumed to be served in FIFO order (FIFO-per-microflow).

For the example scenario in Figure 1, the SFA delay bound for flow 1 can be computed as follows

$$d^{SFA} = h\left(\alpha_1, [\beta_1 - \alpha_2]^+ \otimes [\beta_2 - (\alpha_2 \oslash \beta_1)]^+\right) \tag{1}$$

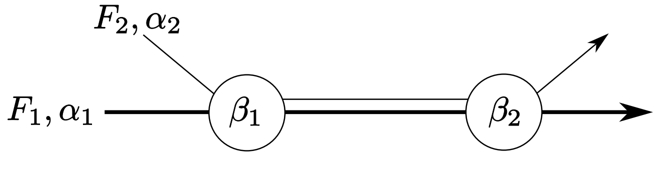

Fig. 1. Simple two nodes, two flows scenario.

TFA vs. SFA In comparison between TFA and SFA, SFA is a clear winner due to the application of the concatenation theorem. Let us illustrate this fact by setting $\alpha_i = \gamma_{r_i,b_i}$ and $\beta_i = \beta_{R_i,T_i}$ for $i = 1, 2$ in the example scenario of Figure 1. We obtain the following delay bounds

$$d^{TFA} = T_1 + T_2 + \tfrac{b_1+b_2+(r_1+r_2)T_1}{R_1-r_1-r_2} + \tfrac{b_1+b_2+(r_1+r_2)(T_1+T_2)}{R_2-r_1-r_2}$$

$$d^{SFA} = T_1 + T_2 + \tfrac{b_1}{(R_1\wedge R_2)-r_2} + \tfrac{b_2+r_2T_1}{R_1-r_2} + \tfrac{b_2+r_2(T_1+T_2)}{R_2-r_2}$$

It can be easily checked that $d^{TFA} \geq d^{SFA}$ is always true and thus the use of SFA is strictly superior to TFA. However, taking a closer look at Equation (1) it becomes clear that d^{SFA} cannot be a tight delay bound since the service at node 1 would be required to be lazy and infinitely fast after the latency at the same time, which can obviously not be the case for any sample path of the system. This observation is also exhibited in the fact that the burst term b_2 appears twice in d^{SFA}, i.e. multiplexing with the interfering flow is *paid twice*. However, the flow of interest cannot be overtaken by the interfering flow's burst twice.

Pay Multiplexing Only Once SFA (PMOO-SFA) A problem with the SFA is the order in which it applies the arbitrary multiplexing theorem and the min-plus convolution. In fact, there is some choice: in the example scenario from Figure 1, we could also first take the convolution of the two nodal service curves and afterwards apply the arbitrary multiplexing to the resulting single node system. In general, the idea is to concatenate single systems first, in order to be able to *pay for the multiplexing with interfering flows only once*, which is why this method is called pay multiplexing only once SFA (PMOO-SFA).

Specifically in the example scenario from Figure 1, a delay bound can be calculated as

$$d^{PMOO} = h\left(\alpha_1, [(\beta_1 \otimes \beta_2 - \alpha_2)]^+\right)$$

Instantiating $\alpha_i = \gamma_{r_i,b_i}$ and $\beta_i = \beta_{R_i,T_i}$ for $i = 1, 2$ in the example scenario, we obtain

$$d^{PMOO} = T_1 + T_2 + \tfrac{b_1+b_2+r_2(T_1+T_2)}{(R_1\wedge R_2)-r_2} \qquad (2)$$

So, as can be observed, based on a clever application order of convolution and the arbitrary multiplexing result, the multiplexing with flow 2 is paid only once (b_2 appears only once). Actually, d^{PMOO} has an intuitive form, with the sum of the nodal latencies and each burst term as well as the burstiness increases at each of the nodes paid at the minimum residual rate of the nodes. So, PMOO-SFA seems to constitute a nice application of the basic network calculus results. In fact, we provided a generalization of this method in [19], where the main issue is dealing with overlapping interference scenarios as depicted for the canonical example in Figure 2 (on page 7).

3.2 Network Calculus Crisis: Convolution Considered Harmful

So far, everything seems to turn out well. However, taking a closer look at the delay bounds for the example scenario for SFA and PMOO-SFA reveals a startling phenomenon: SFA can outperform PMOO-SFA! In particular, let $b_2 = 0, T_1 = 0$ then we obtain

$$\begin{aligned} d^{SFA} &= T_2 + \tfrac{b_1}{(R_1\wedge R_2)-r_2} + \tfrac{r_2T_2}{R_2-r_2} \\ d^{PMOO} &= T_2 + \tfrac{b_1+r_2T_2}{(R_1\wedge R_2)-r_2} \end{aligned}$$

which means that the difference

$$d^{PMOO} - d^{SFA} = r_2T_2\left(\tfrac{1}{(R_1\wedge R_2)-r_2} - \tfrac{1}{R_2-r_2}\right) \geq 0,$$

i.e. the improvement of SFA over PMOO-SFA, in this special case, can be made arbitrarily large by e.g. increasing the rate at the second server R_2. While it is only a special case and in many cases the PMOO-SFA is superior to the SFA (set for example $T_2 = 0$ and the PMOO-SFA always outperforms the SFA), it nevertheless shows that the PMOO-SFA cannot give a tight delay bound in all scenarios. So the question is what goes wrong here, why can we not find a tight bound for this simple scenario by the application of basic network calculus results?

With the SFA, it is clear as discussed above that it does not give a tight bound because it does not correspond to a realizable sample path of the system. Yet, the PMOO-SFA seems like a perfect application of network calculus. In fact, we encounter in this seemingly simple and innocent looking scenario a case where the application of the min-plus convolution is detrimental with respect to finding tight bounds. This can be explained physically when carefully examining the PMOO-SFA delay bound in Equation (2): the burstiness increase of flow 2 due to the latency of node 2, r_2T_2, is paid at the minimum of the residual rates of node 1 and 2. However, that burstiness increase can only be experienced at node 2 and never at node 1, so the PMOO-SFA delay bound cannot capture this physical reality. The convolution effectively "swallows" the topological information that the burstiness increase due to node 2 can only be paid at node 2 (or subsequent nodes in larger scenarios), but not at node 1. The min-plus convolution necessarily is blind for such topological details because it is by definition commutative and thus the order of nodes cannot matter to it when concatenating them.

In this simple example, a very fundamental problem with the direct application of basic network calculus results is exhibited, because the algebraic structure of network calculus being a dioid breaks down here since commutativity is lost. Any aggregate multiplexing that is non-FIFO, even if more knowledge than the arbitrary multiplexing assumption is given, runs into this problem and can thus not be dealt with in the conventional framework of network calculus. The physical reason for this is the following: Without FIFO, there is the possibility at any given node that data from interfering flows which the flow of interest has not yet encountered on previous nodes will overtake the flow of interest, and will thus result in a burstiness increase of the interfering traffic. This burstiness increase must be accounted for at servers downstream starting from the point where it overtook the flow of interest but not on servers upstream from this point.

4 Optimization-Based Bounding Method

In this section, we present an alternative method to compute delay bounds in feed-forward networks of arbitrary multiplexing nodes. This method consists of formulating an optimization problem based on the knowledge about arrival constraints of interfering traffic flows and service guarantees provided by each node. Since the formulation of the general feed-forward network case is notationally heavy, we first present the method for the canonical example of overlapping interference displayed in Figure 2. In fact, this example captures all of the main difficulties for the application of the optimization-based bounding method. The optimization problem is set up under general arrival and strict service curves and solved for the case of token-bucket constrained flows and rate-latency servers. Next we demonstrate that the solution to the optimization problem arrives at a *tight* bound by providing a sample path in which the delay bound is actually experienced. After that, the general feed-forward network case is discussed. At the end of this section, we prove a result on how to compute a left-over

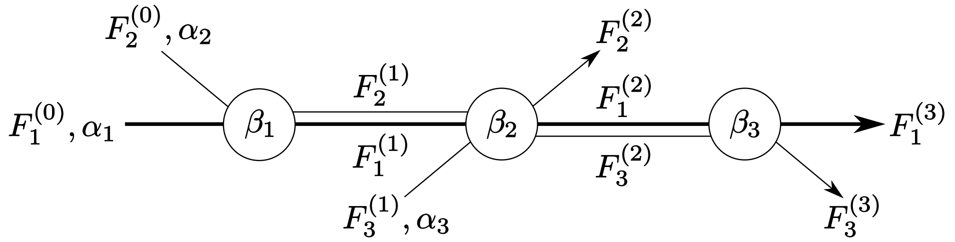

Fig. 2. Overlapping interference scenario.

service curve under piece-wise linear concave arrival and convex service curves, a very relevant case in practice.

4.1 The Case of Overlapping Interference

Let us assume the canonical example scenario of overlapping interference from Figure 2 with the respective denotations. Furthermore, assume that the nodes provide strict service curves $\beta_k, k = 1, 2, 3$.

Assume $0 \leq t_0 \leq t_1 \leq t_2 \leq t_3$ such that t_{k-1} is the start of the last backlogged period at node k before t_k. From the strict service curve property at each of the nodes it is ensured that for $k = 1, 2, 3$

$$F^{(k)}(t_k) - F^{(k-1)}(t_{k-1}) \geq \beta_k(t_k - t_{k-1})$$

where $F^{(k)}$ denotes the total flow entering node $k+1$. Due to the wide-sense increasing nature of the input and output functions and since $F_i^{(k)}(t_k) \geq F_i^{(k)}(t_{k-1}) = F_i^{(k-1)}(t_{k-1})$ for the selected t_k such that $F_i^{(k)}(t_k) - F_i^{(k-1)}(t_{k-1}) \geq 0$ this can be rewritten as

$$F_1^{(3)}(t_3) - F_1^{(2)}(t_2) \geq \left[\beta_3(t_3 - t_2) - \left(F_3^{(3)}(t_3) - F_3^{(2)}(t_2)\right)\right]^+$$

$$F_1^{(2)}(t_2) - F_1^{(1)}(t_1) \geq \left[\beta_2(t_2 - t_1) - \left(F_3^{(2)}(t_2) - F_3^{(1)}(t_1)\right) - \left(F_2^{(2)}(t_2) - F_2^{(1)}(t_1)\right)\right]^+$$

$$F_1^{(1)}(t_1) - F_1^{(0)}(t_0) \geq \left[\beta_1(t_1 - t_0) - \left(F_2^{(1)}(t_1) - F_2^{(0)}(t_0)\right)\right]^+$$

These can be added up so that

$$\begin{aligned}
&F_1^{(3)}(t_3) - F_1^{(0)}(t_0) \\
&\geq \left[\beta_3(t_3 - t_2) - \left(F_3^{(3)}(t_3) - F_3^{(2)}(t_2)\right)\right]^+ \\
&\quad + \left[\beta_2(t_2 - t_1) - \left(F_3^{(2)}(t_2) - F_3^{(1)}(t_1)\right) - \left(F_2^{(2)}(t_2) - F_2^{(1)}(t_1)\right)\right]^+ \\
&\quad + \left[\beta_1(t_1 - t_0) - \left(F_2^{(1)}(t_1) - F_2^{(0)}(t_0)\right)\right]^+
\end{aligned}$$

From the arrival constraints we know that

$$\begin{aligned}
F_2^{(1)}(t_1) - F_2^{(0)}(t_0) &\leq F_2^{(0)}(t_1) - F_2^{(0)}(t_0) \leq \alpha_2(t_1 - t_0) \\
F_2^{(2)}(t_2) - F_2^{(0)}(t_0) &\leq F_2^{(0)}(t_2) - F_2^{(0)}(t_0) \leq \alpha_2(t_2 - t_0) \\
F_3^{(2)}(t_2) - F_3^{(1)}(t_1) &\leq F_3^{(1)}(t_2) - F_3^{(1)}(t_1) \leq \alpha_3(t_2 - t_1) \\
F_3^{(3)}(t_3) - F_3^{(1)}(t_1) &\leq F_3^{(1)}(t_3) - F_3^{(1)}(t_1) \leq \alpha_3(t_3 - t_1)
\end{aligned}$$

Now comes an important step: we introduce slack variables in order to rewrite the previous inequations as

$$\begin{aligned}
F_2^{(1)}(t_1) - F_2^{(0)}(t_0) &= \alpha_2(t_1 - t_0) - s_2^{(1)}, s_2^{(1)} \geq 0 \\
F_2^{(2)}(t_2) - F_2^{(1)}(t_1) &\leq \alpha_2(t_2 - t_0) - \alpha_2(t_1 - t_0) + s_2^{(1)} \\
F_3^{(2)}(t_2) - F_3^{(1)}(t_1) &= \alpha_3(t_2 - t_1) - s_3^{(2)}, s_3^{(2)} \geq 0 \\
F_3^{(3)}(t_3) - F_3^{(2)}(t_2) &\leq \alpha_3(t_3 - t_1) - \alpha_3(t_2 - t_1) + s_3^{(2)}
\end{aligned}$$

The slack variables $s_i^{(k)}$ have the following interpretation: they account for how much of the accumulated burstiness of the respective flow i is transferred from node k to the next node. As such they will be the decision variables for the optimization problem to be solved, because the whole bounding problem is about where each flow's burstiness is to be paid. Next, we need to also bound the slack variables from above with the help of the service curve guarantees. In particular we know for flow 2 that

$$\alpha_2(t_1 - t_0) - s_2^{(1)} = F_2^{(1)}(t_1) - F_2^{(0)}(t_0) \geq \beta_1(t_1 - t_0)$$

from which follows that

$$\begin{aligned} s_2^{(1)} &\leq \alpha_2(t_1 - t_0) - \beta_1(t_1 - t_0) \\ &\leq \sup_{t_1 - t_0 \geq 0} \{\alpha_2(t_1 - t_0) - \beta_1(t_1 - t_0)\} =: B_2^{(1)} \end{aligned}$$

With flow 3 the argument is a bit more complex, we know

$$\begin{aligned} &\alpha_3(t_2 - t_1) - s_3^{(2)} = F_3^{(2)}(t_2) - F_3^{(1)}(t_1) \\ &\geq \left[\beta_2(t_2 - t_1) - \left(F_2^{(2)}(t_2) - F_2^{(1)}(t_1)\right)\right]^+ \\ &\geq \left[\beta_2(t_2 - t_1) - \left(\alpha_2(t_2 - t_0) - \alpha_2(t_1 - t_0) + s_2^{(1)}\right)\right]^+ \end{aligned}$$

such that

$$\begin{aligned} s_3^{(2)} &\leq \alpha_3(t_2 - t_1) - \left[\beta_2(t_2 - t_1) - \left(\alpha_2(t_2 - t_0) - \alpha_2(t_1 - t_0) + s_2^{(1)}\right)\right]^+ \\ &\leq \sup_{t_2 - t_1 \geq 0} \left\{\alpha_3(t_2 - t_1) - \left[\beta_2(t_2 - t_1) - \left(\alpha_2(t_2 - t_0) - \alpha_2(t_1 - t_0) + s_2^{(1)}\right)\right]^+\right\} =: B_3^{(2)} \end{aligned}$$

Here it is important to realize that we give strict priority to flow 2 over flow 3. This is due to the observation that doing so makes flow 3 as bursty as possible and since it still accompanies the flow of interest on node 3 whereas flow 2 leaves after node 2 this constitutes the worst case for the flow of interest. This observation is true in general and thus the scheduling among interfering flows is to always give priority to those flows that leave the flow of interest soonest.

Taking it all together and using the following two logical conditions, $\mathcal{C}_1 = (0 \leq t_0 \leq t_1 \leq t_2 \leq t_3)$ and $\mathcal{C}_2 = \left(0 \leq s_2^{(1)} \leq B_2^{(1)}, 0 \leq s_3^{(2)} \leq B_3^{(2)}\right)$, we arrive at

$$\begin{aligned} &F_1^{(3)}(t_3) - F_1^{(0)}(t_0) \geq \beta^1(t_3 - t_0) \\ &= \inf_{\mathcal{C}_1, \mathcal{C}_2} \Bigg\{ \left[\beta_3(t_3 - t_2) - \left(\alpha_3(t_3 - t_1) - \alpha_3(t_2 - t_1) + s_3^{(2)}\right)\right]^+ \\ &\qquad + \left[\beta_2(t_2 - t_1) - \left(\alpha_3(t_2 - t_1) - s_3^{(2)}\right) - \left(\alpha_2(t_2 - t_0) - \alpha_2(t_1 - t_0) + s_2^{(1)}\right)\right]^+ \\ &\qquad + \left[\beta_1(t_1 - t_0) - \left(\alpha_2(t_1 - t_0) - s_2^{(1)}\right)\right]^+ \Bigg\} \end{aligned}$$

Thus we have to solve a constrained optimization problem in order to find the left-over service curve of flow 1 with which we can then compute the delay bound according to Theorem 1 in the conventional manner. The optimization problem cannot be solved in the general setting assumed so far. When setting $\beta_i = \beta_{R_i,T_i}$ and $\alpha_i = \gamma_{r_i,b_i}$, $i = 1, 2, 3$, we can proceed as follows:

$$B_2^{(1)} = b_2 + r_2 T_1, B_3^{(2)} = b_3 + r_3 \left(T_2 + \frac{s_2^{(1)} + r_2 T_2}{R_2 - r_2}\right)$$

$$\begin{aligned} &\beta^1(t_3 - t_0) \\ &= \inf_{\mathcal{C}_1, \mathcal{C}_2} \Bigg\{ \left[R_3 [t_3 - t_2 - T_3]^+ - \left(r_3(t_3 - t_2) + s_3^{(2)}\right)\right]^+ \\ &\qquad + \left[R_2 [t_2 - t_1 - T_2]^+ - \left(b_3 + r_3(t_2 - t_1) - s_3^{(2)}\right) - \left(r_2(t_2 - t_1) + s_2^{(1)}\right)\right]^+ \\ &\qquad + \left[R_1 [t_1 - t_0 - T_1]^+ - \left(b_2 + r_2(t_1 - t_0) - s_2^{(1)}\right)\right]^+ \Bigg\} \end{aligned}$$

$$
\begin{aligned}
&\geq \inf_{\mathcal{C}_1,\mathcal{C}_2}\Bigg\{(R_3-r_3)\left[t_3-t_2-T_3-\tfrac{r_3T_3+s_3^{(2)}}{R_3-r_3}\right]^+\\
&\qquad+(R_2-r_2-r_3)\left[t_2-t_1-T_2-\tfrac{b_3+(r_2+r_3)T_2-s_3^{(2)}+s_2^{(1)}}{R_2-r_2-r_3}\right]^+\\
&\qquad+(R_1-r_2)\left[t_1-t_0-T_1-\tfrac{b_2+r_2T_1-s_2^{(1)}}{R_1-r_2}\right]^+\Bigg\}\\
&=((R_1-r_2)\wedge(R_2-r_2-r_3)\wedge(R_3-r_3))\times\\
&\quad\inf_{\mathcal{C}_2}\left\{\left[t_3-t_0-(T_1+T_2+T_3)-\tfrac{r_3T_3+s_3^{(2)}}{R_3-r_3}-\tfrac{b_3+(r_2+r_3)T_2-s_3^{(2)}+s_2^{(1)}}{R_2-r_2-r_3}-\tfrac{b_2+r_2T_1-s_2^{(1)}}{R_1-r_2}\right]^+\right\}
\end{aligned}
$$

Hence, to find a closed form for the left-over service curve is equivalent to solving the following optimization problem:

$$
\begin{aligned}
min.\;&\left(\tfrac{1}{R_1-r_2}-\tfrac{1}{R_2-r_2-r_3}\right)s_2^{(1)}+\left(\tfrac{1}{R_2-r_2-r_3}-\tfrac{1}{R_3-r_3}\right)s_3^{(2)}\\
s.t.\;&0\leq s_2^{(1)}\leq b_2+r_2T_1\\
&0\leq s_3^{(2)}\leq b_3+r_3\left(T_2+\tfrac{s_2^{(1)}+r_2T_2}{R_2-r_2}\right)
\end{aligned}
$$

This constitutes a simple linear program that can be solved easily when the parameters are given. The general solution of this problem depends on the relations between the residual rates at the nodes. With $A:=\frac{1}{R_1-r_2}-\frac{1}{R_2-r_2-r_3}$, $B:=\frac{1}{R_2-r_2-r_3}-\frac{1}{R_3-r_3}$, and $C:=\frac{r_3}{R_2-r_2}$ the solution vector for the slack variables $\left(s_2^{(1)},s_3^{(2)}\right)$ is obtained as

$$
\begin{aligned}
&A\leq 0,B>0:(b_2+r_2T_2,0)\\
&A\leq 0,B\leq 0:\left(b_2+r_2T_1,b_3+r_3\left(T_2+\tfrac{b_2+r_2T_1+r_2T_2}{R_2-r_2}\right)\right)\\
&A>0,B>0:(0,0)\\
&\begin{matrix}A>0,B\leq 0,\\ CB\leq A\end{matrix}:\left(0,b_3+r_3\left(T_2+\tfrac{r_2T_2}{R_2-r_2}\right)\right)\\
&\begin{matrix}A>0,B\leq 0,\\ CB> A\end{matrix}:\left(b_2+r_2T_1,b_3+r_3\left(T_2+\tfrac{b_2+r_2T_1+r_2T_2}{R_2-r_2}\right)\right)
\end{aligned}
$$

From this the left-over (l.o.) service curve for flow 1 can be given. We state it for the last case ($A>0,B\leq 0,CB>A$):

$$\beta^1=\beta_{R^{l.o.},T^{l.o.}} \tag{3}$$

with

$$
\begin{aligned}
R^{l.o.}&=(R_1-r_2)\wedge(R_2-r_2-r_3)\wedge(R_3-r_3)\\
T^{l.o.}&=T_1+T_2+T_3+\tfrac{b_3+r_3(T_2+T_3)}{R_3-r_3}+\tfrac{b_2+r_2(T_1+T_2)}{R_2-r_2-r_3}+\tfrac{(R_2-r_2-R_3)r_3\frac{b_2+r_2(T_1+T_2)}{R_2-r_2}}{(R_2-r_2-r_3)(R_3-r_3)}
\end{aligned}
$$

This case corresponds to a situation where $R_1-r_2<R_2-r_2-r_3$ ($A>0$) and $R_2-r_2-r_3\geq R_3-r_3$ ($B\leq 0$) and $r_2>\frac{(R_2-R_1)R_3}{R_2-r_2}$ ($CB>A$). It is interesting because it constitutes a case where the burst of flow 2 is not paid at node 1, although node 1's residual service is slower than that for node 2. Instead it is paid at node 2, because its effect of increasing the burstiness of flow 3, which is then paid at node 3, is stronger than if it had been paid at node 1. This shows that for some scenarios the delay bound can depend on fairly contrived conditions on the parameters of the scenario.

Setting $F_3=0$ and $\beta_3=+\infty$ in the overlapping interference scenario results in the simple example scenario from Section 3. Here, the left-over service curve for flow 1 and the corresponding delay bound are

$$\beta^1=\beta_{(R_1\wedge R_2)-r_2,T_1+T_2+\frac{b_2+r_2T_1}{(R_1\wedge R_2)-r_2}+\frac{r_2T_2}{R_2-r_2}}$$

$$d^{OPT} = h\left(\alpha_1, \beta^1\right) = T_1 + T_2 + \frac{b_1+b_2+r_2T_1}{(R_1 \wedge R_2)-r_2} + \frac{r_2T_2}{R_2-r_2}$$

The burstiness increase due to node 2 (r_2T_2) is now correctly accounted for at node 2 instead of node 1 as for the PMOO-SFA. At the same time the burst terms are only accounted for once, thus the PMOO gain is still retained.

4.2 Tightness of Delay Bound

Using the optimization-based method to find the left-over service curve for a flow of interest actually achieves a tight delay bound. Let us illustrate this for the interesting case treated in the previous subsection ($A > 0, B \leq 0, CB > A$). The other cases can be treated similarly, but are simpler. Assume the left-over service curve in Equation (3) is tight, i.e. it is the largest service curve (for all interval lengths) that can be given to the flow of interest. Then the delay bound can be easily seen to be tight from the tightness of the delay bound in a simple single node system as mentioned in Section 2 for Theorem 1. Hence, it has to be shown that the left-over service curve in Equation (3) is tight. First we show the tightness of the latency, i.e. there is actually a sample path such that the system offers no service to the flow of interest for the duration of the latency:

We track a specific bit of flow 1 (the flow of interest), called the bit-under-observation (b-u-o), which arrives at node 1 at time t_0, and create a worst-case sample path for the time until this bit leaves node 3. This constitutes the latency for the flow of interest.

All interfering flows are assumed to be constantly claiming their sustained rates r_i, however when they burst will be detailed below. Similarly, when the servers take their latencies is given below. Note that not all the servers are lazy, i.e. some provide more than their service curves ensure (see 2) below). In fact, otherwise the worst-case sample path with respect to the latency would not be attained.

1. At time t_0: node 1 takes its latency T_1; flow 2 bursts.
2. At time $t_1 = t_0 + T_1$: node 1 starts serving infinitely fast, thus the backlog of flow 2, $b_2 + r_2T_1$, and the b-u-o are passed on to node 2; node 2 immediately takes its latency; flow 3 bursts.
3. At time $t_2 = t_1 + T_2$: node 2 starts serving flow 2 at the minimal rate R_2 with strict priority over flow 3; flow 2 has a backlog at node 2 of $x_2^{(2)}(t_2) = b_2 + r_2(T_1 + T_2)$.
4. At time $t_3 = t_2 + \frac{x_2^{(2)}(t_2)}{R_2-r_2}$: flow 3 starts to be served by node 2 with strict priority over flow 1; flow 3 has a backlog at node 2 of $x_3^{(2)}(t_3) = b_3 + r_3T_2 + r_3\frac{x_2^{(2)}(t_2)}{R_2-r_2}$; at the same time bits from flow 2 arrive at rate r_2 and are still served with priority over flow 3, hence reducing the service rate for flow 3 to $R_2 - r_2$.
5. At time $t_4 = t_3 + \frac{x_3^{(2)}(t_3)}{R_2-r_2-r_3}$: the b-u-o leaves node 2 and is passed on to node 3; node 3 takes its latency; flow 3 has a backlog at node 3 of $x_3^{(3)}(t_4) = (R_2 - r_2 - R_3)\frac{x_3^{(2)}(t_3)}{R_2-r_2-r_3}$. Note that $B \leq 0$ implies $R_2 - r_2 \geq R_3$ and thus a backlog for flow 3 builds up at node 3.
6. At time $t_5 = t_4 + T_3$: flow 3 is continued to be served by node 3 with strict priority over flow 1; flow 3 has a backlog of $x_3^{(3)}(t_5) = r_3T_3 + x_3^{(3)}(t_4)$.
7. At time $t_6 = t_5 + \frac{x_3^{(3)}(t_5)}{R_3-r_3}$: the b-u-o leaves node 3.

This sample path is also illustrated in Figure 3. The latency for the b-u-o can thus be computed as

$$t_6 - t_0 = T_1 + T_2 + T_3 + \frac{b_2+r_2(T_1+T_2)}{R_2-r_2} + \frac{b_3+r_3T_2+r_3\frac{b_2+r_2(T_1+T_2)}{R_2-r_2}}{R_2-r_2-r_3} + \frac{r_3T_3+(R_2-r_2-R_3)\frac{b_3+r_3T_2+r_3\frac{b_2+r_2(T_1+T_2)}{R_2-r_2}}{R_2-r_2-r_3}}{R_3-r_3}$$

This expression can be checked to be equal to $T^{l.o.}$ and thus the latency of the left-over service curve is shown to be tight because it is attained for the above described sample path.

For a time interval longer than the latency, $t > T^{l.o.}$, the service curve can be seen to be tight by considering the following sample path: During a period of length $t - T^{l.o.}$ each interfering flow creates traffic according to its sustained rate. This is followed by a period of length $T^{l.o.}$ for which the sample path is the one described above to illustrate the tightness of the latency. Note that it is important to take the outage time of the service for the flow of interest (the latency of the left-over service curve) at the end of the interval. Otherwise, if for example the last server is not the bottleneck server, this server could play out buffered data faster than at the minimum residual rate over all nodes, resulting in a seemingly larger service curve. Yet, this would not constitute the worst-case sample path.

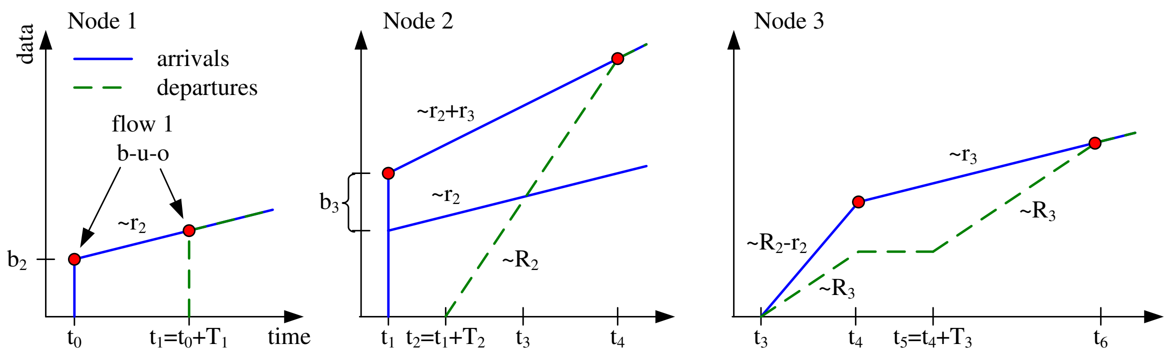

Fig. 3. Worst-case sample path illustration.

4.3 General Feed-forward Networks

In this subsection, the extension of the optimization-based bounding method to general feed-forward networks is discussed. We essentially follow the same steps as for the overlapping interference scenario in Section 4.1.

A general feed-forward network scenario looks like Figure 4 from the perspective of the flow of interest. So there are potentially $\frac{n}{2}(n+1)$ interfering flows, one for each possible joining node i and leaving node j, denoted by $F_{i,j}^{(k)}, k = i-1, ..., j$, where the upper index denotes the node where the flow is coming from (for a newly joining flow at node i it is set to $i-1$, for ease of generalization). In the following we assume all $\frac{n}{2}(n+1)$ possible interfering flows to exist, if that is not the case we can simply set those that are missing to 0.

The flow of interest is denoted by $F_{int}^{(k)}$, $k = 0, ..., n$. Assume $t_0 \le t_1 \le t_2 \le ... \le t_n$, such that t_{i-1} is the start of the last backlogged period at node i before t_i. Due to the strict service curve property at each of the nodes and following the same arguments about the wide-sense increasing nature of input and output functions as well as the choice of the selected t_i, we obtain the following for the flow of interest

$$F_{int}^{(n)}(t_n) - F_{int}^{(0)}(t_0) \ge \sum_{k=1}^{n} \left[\beta_k(t_k - t_{k-1}) - \sum_{l,m:l\le k\le m} \left(F_{l,m}^{(k)}(t_k) - F_{l,m}^{(k-1)}(t_{k-1}) \right) \right]^+$$

From the causality of the system and the arrival constraints for each of the interfering flows $F_{i,j}^{(k)}$ we know that for $k : i \le k \le j$

$$F_{i,j}^{(k)}(t_k) - F_{i,j}^{(i-1)}(t_{i-1}) \le F_{i,j}^{(i-1)}(t_k) - F_{i,j}^{(i-1)}(t_{i-1}) \le \alpha_{i,j}(t_k - t_{i-1})$$

Introducing again slack variables for each interfering flow and each of its traversed nodes, $s_{i,j}^{(k)} \ge 0$, these inequations can be rewritten as (for $i, j : 1 \le i \le j \le n$, $k : i-1 \le k \le j$)

$$F_{i,j}^{(k)}(t_k) - F_{i,j}^{(k-1)}(t_{k-1}) = \alpha_{i,j}(t_k - t_{i-1}) - \left(\alpha_{i,j}(t_{k-1} - t_{i-1}) - s_{i,j}^{(k-1)} \right) - s_{i,j}^{(k)} \qquad (4)$$

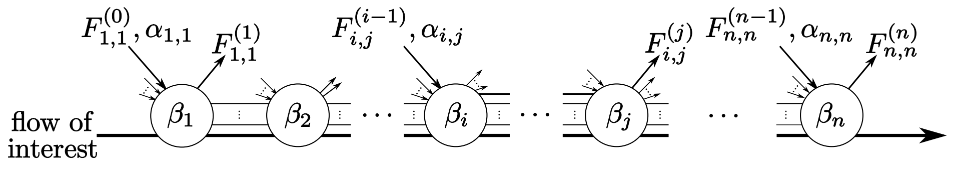

Fig. 4. General feed-forward network scenario.

To be correct, note that for $k = j$, the equation sign must be replaced by $\geq$. Further note that we set $s_{i,j}^{(j)} := 0$ and $s_{i,j}^{(i-1)} := \alpha_{i,j}(0)$.

Next the upper bound on the slack variables is derived

$$\alpha_{i,j}(t_k - t_{i-1}) - \left(\alpha_{i,j}(t_{k-1} - t_{i-1}) - s_{i,j}^{(k-1)}\right) - s_{i,j}^{(k)} = F_{i,j}^{(k)}(t_k) - F_{i,j}^{(k-1)}(t_{k-1})$$

$$\geq \left[\beta_k(t_k - t_{k-1}) - \sum_{\substack{l,m\,:\,l \leq k \leq m,\\ (m<j) \vee (m=j \wedge i>l)}} \left(F_{l,m}^{(k)}(t_k) - F_{l,m}^{(k-1)}(t_{k-1})\right)\right]^+$$

Here, a strict priority is given again to those flows that leave the flow of interest soonest, ties are broken arbitrarily based on the joining node of the interfering flows. Using Equation (4) we obtain the following as an upper bound on the slack variables

$$s_{i,j}^{(k)} \leq \sup_{t_k - t_{k-1} \geq 0} \left\{ \alpha_{i,j}(t_k - t_{i-1}) - \left(\alpha_{i,j}(t_{k-1} - t_{i-1}) - s_{i,j}^{(k-1)}\right) - \left[\beta_k(t_k - t_{k-1}) \right.\right.$$

$$\left.\left. - \sum_{\substack{l,m\,:\,l \leq k \leq m,\\ (m<j) \vee (m=j \wedge i>l)}} \left(\alpha_{l,m}(t_k - t_{l-1}) - \left(\alpha_{l,m}(t_{k-1} - t_{l-1}) - s_{l,m}^{(k-1)}\right) - s_{l,m}^{(k)}\right)\right]^+ \right\} =: B_{i,j}^{(k)}$$

Taking it all together and using the following logical conditions, $\mathcal{C}_1 = (1 \leq i \leq n) \wedge (t_i \geq t_{i-1})$ and $\mathcal{C}_2 = (1 \leq i \leq k \leq j \leq n) \wedge \left(0 \leq s_{i,j}^{(k)} \leq B_{i,j}^{(k)}\right)$, we arrive at

$$F_{int}^{(n)}(t_n) - F_{int}^{(0)}(t_0) \geq \inf_{\mathcal{C}_1, \mathcal{C}_2} \left\{ \sum_{k=1}^{n} \left[\beta_k(t_k - t_{k-1}) \right.\right.$$

$$\left.\left. - \sum_{l,m:l \leq k \leq m} \left(\alpha_{l,m}(t_k - t_{l-1}) - \left(\alpha_{l,m}(t_{k-1} - t_{l-1}) - s_{l,m}^{(k-1)}\right) - s_{l,m}^{(k)}\right)\right]^+ \right\}$$

Hence, to find the left-over service curve, we have to solve again a constrained optimization problem. If token-bucket constrained interfering flows and rate-latency servers are assumed, it is of a linear form again. The number of decision variables is the number of interfering flows times their respective number of nodes for which they accompany the flow of interest. At a maximum the number of decision variables can become

$$\sum_{i=1}^{n} \sum_{j=i}^{n} (j - i + 1) = \frac{n(n+1)(n+2)}{6} = \mathcal{O}\left(n^3\right).$$

The number of constraints is two times the number of decision variables (one upper and lower bound for each decision variable). So, if the concrete parameter values of the token buckets and rate-latency curves are given, standard methods as for example the simplex algorithm can be used to solve such a problem. In general, however, there is not much hope for arriving at a closed-form solution without an excessive amount of case discriminations, unless the optimization problem has a special form. Therefore, we stop the derivation here and come back to it in Section 5, where we treat the case of sink trees, which results in such a special form so that the optimization problem can be solved explicitly again.

4.4 Generalization to Piecewise Linear Curves

Based on the method presented in the previous subsections, we can compute delay bounds for the case of token-bucket constrained interfering flows and rate-latency servers. Now we provide a result that allows to generalize that method to the frequent case of piecewise linear concave arrival and convex service curves.

Proposition 1. *Given piecewise linear concave arrival curves* $\alpha_i = \bigwedge_{k_i=1}^{n_i} \gamma_{r_{k_i},b_{k_i}}$ *for each interfering flow* $i = 1,\ldots,n$ *and piecewise linear convex strict service curves* $\beta_j = \bigvee_{l_j=1}^{m_j} \beta_{R_{l_j},T_{l_j}}$ *for each node* $j = 1,\ldots,m$ *on the path of the flow of interest, the left-over service curve for the flow of interest under arbitrary multiplexing is given by*

$$\beta^{l.o.} \geq \bigvee_{i=1}^{n} \bigvee_{j=1}^{m} \bigvee_{k_i=1}^{n_i} \bigvee_{l_j=1}^{m_j} \beta^{l.o.}_{\{k_i\},\{l_i\}} \tag{5}$$

where $\beta^{l.o.}_{\{k_i\},\{l_j\}}$ *are end-to-end left-over service curves under arbitrary multiplexing for a specific combination of a single token bucket per interfering flow and a single rate-latency curve per node.*

Proof. Note that we can also view left-over service curves as functions $\beta^{l.o.}(\boldsymbol{\alpha},\boldsymbol{\beta})$ of the arrival curves of the interfering flows ($\boldsymbol{\alpha} = (\alpha_1,...,\alpha_n)$) as well as the service curves of the nodes ($\boldsymbol{\beta} = (\beta_1,...,\beta_m)$ on the path of the flow of interest. It should be clear that left-over service curves are wide-sense decreasing in their arrival curve arguments and wide-sense increasing in their service curve arguments. That means, if any of the arrival curves α_i is substituted by a (strictly) larger one, then the left-over service curve can only become smaller. For the service curve arguments β_i, if any becomes larger, then the left-over service curve can only become larger, too.

Now we prove by a combination of structural induction on the given scenario and contradiction:

Induction start: For $n_i = m_j = 1; i = 1,\ldots,n; j = 1,\ldots,m$ the statement in the proposition is tautological.

Induction step: Assume the proposition to be correct for given n_i and m_j. Now suppose, at first for $n_i + 1$ Equation (5) is wrong, i.e. $\exists t \geq 0$ such that

$$\beta^{l.o.}\left(\left(\alpha_1,...,\alpha_i \wedge \gamma_{r_{n_{i+1}},b_{n_{i+1}}},...,\alpha_n\right),\boldsymbol{\beta}\right)(t) < \beta^{l.o.}(\boldsymbol{\alpha},\boldsymbol{\beta})(t)$$

or

$$\beta^{l.o.}\left(\left(\alpha_1,...,\alpha_i \wedge \gamma_{r_{n_i+1},b_{n_{i+1}}},...,\alpha_n\right),\boldsymbol{\beta}\right)(t) < \beta^{l.o.}\left(\left(\alpha_1,...,\gamma_{r_{n_i+1},b_{n_{i+1}}},...,\alpha_n\right),\boldsymbol{\beta}\right)(t)$$

None of the two can be the case because they would violate the wide-sense decreasing nature of the left-over service curve for the arrival curve arguments. Since we instantiate strict service curves only at the beginning of backlogged periods, i.e. the respective time instances are identical in all cases, the maximum of all lower bounds that are proven above constitutes a service curve. Thus the proposition is also true for $n_i + 1$. The induction step over the service curve arguments follows along similar lines. $\square$

So, under the assumption of piecewise linear curves we now have to solve a *set* of linear programs and then compute the pointwise maximum of the service curves in this set. This can become pretty compute-intensive if the amount of linear segments used to model arrival and service curves grows. In particular, the number of left-over service curves to be computed and thus the number of linear programs to be solved is $\prod_{i=1}^{n}\prod_{j=1}^{m} n_i m_j$.

5 Explicit Delay Bounds for Sink trees

Using the optimization-based bounding method presented in the previous section, we derive explicit delay bounds for sink-tree networks in this section. Sink trees are a frequent special case of feed-forward networks. For example, MPLS networks use sink-tree aggregation of label switched paths from one edge of a network to the other. Another example are wireless sensor networks, which also often organize their topology as a sink-tree towards a base station collecting the data.

While applying the optimization-based method to general feed-forward networks is well possible if the parameter values of arrival and service curves are given, it is hard to conceive that a closed-form solution for delay bounds under general arrival and service curves can be given. For sink-tree networks this is different: The optimization problem of finding the left-over service curve for a flow of interest

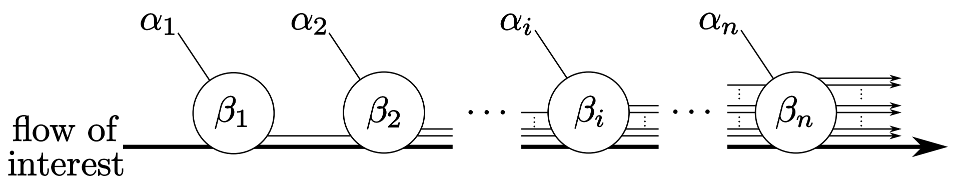

Fig. 5. Sink-tree scenario.

has a form that allows to solve it under a general setting of piecewise linear concave arrival and convex service curves (without instantiation of their parameters). Next we prove the result for computing the left-over service curve. The closed-form expression of the delay bound for a flow of interest then follows simply from the application of Theorem 1 by calculating the horizontal deviation between the flow's arrival curve and its left-over service curve.

Proposition 2. *Assume a sink-tree scenario as illustrated in Figure 5, note that the arrival curves α_i for the interfering flows can be calculated using the output bound from Theorem 1. The left-over service curve for the flow of interest under piecewise linear concave arrival curves $\alpha_i = \bigwedge_{k_i=1}^{n_i} \gamma_{r_{k_i},b_{k_i}}$ and convex strict service curves $\beta_i = \bigvee_{l_i=1}^{m_i} \beta_{R_{l_i},T_{l_i}}$ is given by*

$$\beta^{l.o.} = \bigvee_{i=1}^{n} \bigvee_{k_i=1}^{n_i} \bigvee_{l_i=1}^{m_i} \beta_{R^{l.o.}_{\{k_i\},\{l_i\}},T^{l.o.}_{\{k_i\},\{l_i\}}}$$

with

$$R^{l.o.}_{\{k_i\},\{l_i\}} = \bigwedge_{i=1}^{n} \left(R_{l_i} - \sum_{j=1}^{i} r_{k_j} \right)$$

$$T^{l.o.}_{\{k_i\},\{l_i\}} = \sum_{i=1}^{n} \left(T_i + \frac{b_{k_i}}{\bigwedge_{j=i}^{n} \left(R_{l_j} - \sum_{s=1}^{j} r_{k_s} \right)} + \sum_{j=i}^{n} \frac{r_{k_i} T_{l_j}}{\bigwedge_{s=j}^{n} \left(R_{l_s} - \sum_{v=1}^{s} r_{k_v} \right)} \right)$$

Proof. Using Proposition 1 from Section 4.4, it suffices to show that Proposition 2 is true for the special case with $n_i = m_i = 1$, i.e. the interfering flows are assumed to be token-bucket constrained and the servers are of the rate-latency type. In Section 4.3, we described how to proceed for general feed-forward networks up to the point where arrival and service curves need to be instantiated with a certain shape (not yet their parameters). We now continue at this point for the special case of sink trees. In particular, the optimization problem can be rephrased as: Under the following logical conditions, $\mathcal{C}_1 = (1 \leq i \leq n) \wedge (t_i \geq t_{i-1})$ and $\mathcal{C}_2 = (1 \leq i \leq k \leq n) \wedge \left(0 \leq s_i^{(k)} \leq B_i^{(k)}\right)$,

$$F_{int}^{(n)}(t_n) - F_{int}^{(0)}(t_0) \geq \inf_{\mathcal{C}_1,\mathcal{C}_2} \left\{ \sum_{k=1}^{n} \left[\beta_k (t_k - t_{k-1}) \right.\right.$$

$$\left.\left. - \sum_{l:l \leq k} \left(\alpha_l (t_k - t_{l-1}) - \left(\alpha_l (t_{k-1} - t_{l-1}) - s_l^{(k-1)} \right) - s_l^{(k)} \right) \right]^+ \right\}$$

Note that the second index (the egress points of interfering flows) could be omitted for several variables, because all interfering flows accompany the flow of interest to the destination node n, thus the ingress node is sufficient for identification.

The upper bounds of the $s_i^{(k)}$ can be computed as follows

$$B_i^{(k)} = \sup_{t_k - t_{k-1} \geq 0} \left\{ \alpha_i (t_k - t_{i-1}) - \left(\alpha_i (t_{k-1} - t_{i-1}) - s_i^{(k-1)} \right) - \left[\beta_k (t_k - t_{k-1}) \right.\right.$$

$$- \sum_{\substack{l : l \leq k \\ l < i}} \left(\alpha_l \left(t_k - t_{l-1}\right) - \left(\alpha_l \left(t_{k-1} - t_{l-1}\right) - s_l^{(k-1)}\right) - s_l^{(k)}\right) \Big]^{+} \Bigg\}$$

$$\geq \sup_{t_k - t_{k-1} \geq 0} \left\{ \sum_{l=1}^{i} \left(\alpha_l \left(t_k - t_{l-1}\right) - \alpha_l \left(t_{k-1} - t_{l-1}\right)\right) - \beta_k \left(t_k - t_{k-1}\right) \right\} + s_i^{(k-1)} + \sum_{l=1}^{i-1} \left(s_l^{(k-1)} - s_l^{(k)}\right)$$

$$= V_i^{(k)} + s_i^{(k-1)} + \sum_{l=1}^{i-1} \left(s_l^{(k-1)} - s_l^{(k)}\right)$$

Note that in the summation (within the sup) only the arbitrary tie-breaker remained, because scheduling *between* interfering flows does not matter here. This fact is also the physical explanation why the optimization problem is of a "nicer" form, that allows to arrive at explicit expressions. The $\geq$ step can be done because it means the optimization problem is solved under more stringent constraints, while still arriving at tight bounds, thus showing that the optimal solution has not been excluded from the feasible region.

Next, the actual instantiation with $\alpha_i = \gamma_{r_i, b_i}$ and $\beta_i = \beta_{R_i, T_i}, i = 1, ..., n$ is performed and, following similar lines as in Section 4.1, we obtain

$$F_{int}^{(n)}\left(t_n\right) - F_{int}^{(0)}\left(t_0\right) \geq \bigwedge_{k=1}^{n} \left(R_k - \sum_{l=1}^{k} r_l\right) \inf_{\mathcal{C}_2} \Bigg\{ \Bigg[t_n - t_0 - \sum_{k=1}^{n} T_k$$

$$- \sum_{k=1}^{n} \frac{b_k + \left(\sum_{l=1}^{k} r_l\right) T_k + \sum_{l=1}^{k} \left(s_l^{(k-1)} - s_l^{(k)}\right)}{\bigwedge_{k=1}^{n} \left(R_k - \sum_{l=1}^{k} r_l\right)} \Bigg]^{+} \Bigg\}$$

and

$$V_i^{(k)} = b_i 1_{\{i=k\}} + \left(\sum_{l=1}^{i} r_i\right) T_k$$

This can be put together more clearly as the following linear programming problem

$$\max. \sum_{k=1}^{n} \frac{b_k + \left(\sum_{l=1}^{k} r_l\right) T_k + \sum_{l=1}^{k} \left(s_l^{(k-1)} - s_l^{(k)}\right)}{\bigwedge_{k=1}^{n} \left(R_k - \sum_{l=1}^{k} r_l\right)}$$

$$s.t. \;\; 0 \leq s_i^{(k)} \leq V_i^{(k)} + s_i^{(k-1)} + \sum_{l=1}^{i-1} \left(s_l^{(k-1)} - s_l^{(k)}\right), k = 1, ..., n, \;\; i = 1, ..., k$$

The explicit solution to this problem can be understood by using the structure of the problem as well making oneself the interpretation of the $s_i^{(k)}$ clear: it is the accumulated burstiness of interfering flow i *passed on* from node k upstream. First look at $0 \leq s_1^{(1)} \leq V_1^{(1)} = b_1 + r_1 T_1$: the burstiness due to flow 1 should be passed on to upstream nodes if any of the residual rates of upstream servers is smaller than that of node 1, that means $s_1^{(1)}$ should be set to its upper bound $b_1 + r_1 T_1$. Otherwise (node 1 has the lowest residual rate), it should be set to its lower bound 0, because the multiplexing with flow 1 should then be paid at node 1 (to be more accurate its accumulated burstiness up to that point). Looking now at node 2, we have to decide on the accumulated burstiness due to flow 1 and 2 (since they have last been paid) which is either passed on to upstream nodes or not, meaning they either take on their upper or lower bound. Clearly the decision will be the same for both flows here, because it is again based on the residual rates of node 2 and upstream nodes, which are the same for both flows due to the sink-tree topology. Hence, $s_1^{(2)}$ and $s_2^{(2)}$ are decided. This argument can be continued and results in the fact that any burst term and any burstiness increase is paid at the node with the lowest residual rate on the subpath they actually traverse. □

The tightness of the left-over service curve in the above derivation can be seen using a sample path argument as in Section 4.2. However, the sample path is actually simpler to construct owing to the fact that overlapping interference is avoided in sink trees. Here, we just state the way to construct a worst-case sample path to attain the latency of the left-over service curve (the remaining argumentation follows along the same lines as in Section 4.2): Interfering flows always burst when the bit-under observation of the flow of interest just arrives at the node where the respective interfering flow joins; at the same time instant the respective node takes its latency; otherwise interfering flows send at their sustained rate and servers provide their minimum rate. Hence, the worst-case sample path is fairly straightforward compared to the overlapping interference scenario.

The latency terms of the left-over service curve have an intuitive form: Each interfering flow's burst and its burstiness increases at subsequent nodes are paid at the minimum residual rate server they actually traverse. This insight also means that if residual rates are monotonically decreasing on the path of the flow of interest then PMOO-SFA actually becomes tight. Furthermore, note that the latency terms, and therefore also the delay bound, scale *quadratically* in the number of nodes on the path of the flow of interest.

5.1 Numerical Experiments

Let us now shed some light on what can be the quantitative gain in using the optimization-based bounding method compared to the other methods based on direct application of network calculus for the case of sink-tree networks.

We choose a simple experimental setup with fully occupied binary trees, where each node acts as a source of data. The sources are token-bucket constrained with a sustained rate of 10 Mbps and a bucket depth of 1 Mb. Each node offers a strict rate-latency service curve with a latency of 0.1 ms and the service rate dimensioned such that a certain target utilization is achieved. The server utilizations are varied over the experiments. All calculations have been performed using the DISCO Network Calculator[1].

In the first experiment we compare, under varying tree depths, all of the methods described in the report: TFA, SFA, PMOO-SFA, and the optimization-based bounding method (denoted as TIGHT in the figures). The delay bounds for a leaf in the sink-tree under a utilization of 20% at each of the nodes are displayed in Figure 6.

Clearly, we can observe a considerable improvement of the delay bounds with increasing tree depths. Interestingly, it can also be observed, that the PMOO-SFA, while being second best for tree depths up to 12, is outperformed by SFA and even TFA for larger tree depths. This stresses again the insights gained in Section 3.

In a second experiment, the effect of the utilization on the delay bounds are investigated. Both, the tight bound and the PMOO-SFA are computed for utilizations of 20, 50, and 90%, again for varying tree depths. The results are shown in Figure 7.

As can be observed, the tight bound reacts pretty mildly on higher utilizations, with an approximately linear increase of the delay bound when increasing the tree depth (the quadratic regime mentioned above only shows for even larger tree depths), while the PMOO-SFA suffers badly from higher utilizations and exhibits a steep super-linear increase in its bounds for increasing tree depths. As an extreme example, for a tree depths of 15 at a utilization of 90%, the PMOO-SFA bound is 1382s, while the tight bound is about 6s.

6 Conclusions

In this report, we have demonstrated that direct application of network calculus in aggregate multiplexing scenarios for which FIFO cannot be assumed is problematic with respect to the tightness of delay bounds. The nice algebraic characteristics of network calculus cannot be preserved under arbitrary multiplexing. Based on this insight, we have proposed an alternative method to arrive at tight bounds for arbitrary multiplexing in feed-forward networks. The method consists of formulating

[1] The DISCO Network Calculator is publicly available under `http://disco.informatik.uni-kl.de/content/Downloads`.

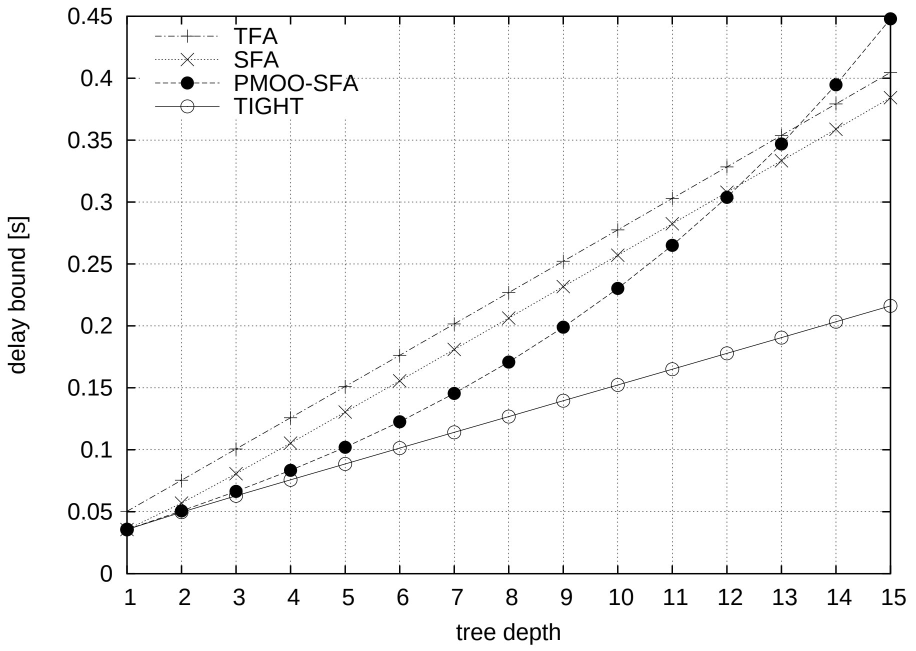

Fig. 6. All bounding methods compared at a utilization of 20%.

an optimization problem. For the case of piecewise linear concave arrival and convex service curves (a very frequent case), we obtain a set of linear programming problems, which can be solved by standard methods, and from which the solution for a tight delay bound can be composed. For the special case of sink-tree networks we have provided a closed-form expression of the delay bound based on this method. Numerical experiments have shown a clear edge for the novel bounding method over methods based on direct application of network calculus.

References

1. R. L. Cruz, "A calculus for network delay, Part I: Network elements in isolation," *IEEE Transactions on Information Theory*, vol. 37, pp. 114–131, Jan. 1991.
2. R. L. Cruz, "A calculus for network delay, Part II: Network analysis," *IEEE Transactions on Information Theory*, vol. 37, pp. 132–141, Jan. 1991.
3. R. L. Cruz, "Quality of service guarantees in virtual circuit switched networks," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 1048–1056, Aug. 1995.
4. H. Sariowan, R. L. Cruz, and G. C. Polyzos, "Scheduling for quality of service guarantees via service curves," in *Proc. IEEE ICCCN*, pp. 512–520, Sept. 1995.
5. C.-S. Chang, "On deterministic traffic regulation and service guarantees: A systematic approach by filtering," *IEEE Transactions on Information Theory*, vol. 44, pp. 1097–1110, May 1998.
6. J.-Y. Le Boudec, "Application of network calculus to guaranteed service networks," *IEEE Transactions on Information Theory*, vol. 44, pp. 1087–1096, May 1998.
7. R. Agrawal, R. L. Cruz, C. Okino, and R. Rajan, "Performance bounds for flow control protocols," *IEEE/ACM Transactions on Networking*, vol. 7, pp. 310–323, June 1999.
8. F. Ciucu, A. Burchard, and J. Liebeherr, "A network service curve approach for the stochastic analysis of networks," in *Proc. ACM SIGMETRICS*, pp. 279–290, June 2005.
9. J.-Y. Le Boudec and P. Thiran, *Network Calculus A Theory of Deterministic Queuing Systems for the Internet.* No. 2050 in Lecture Notes in Computer Science, Berlin, Germany: Springer-Verlag, 2001.
10. F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and Linearity: An Algebra for Discrete Event Systems.* Probability and Mathematical Statistics, John Wiley & Sons Ltd., 1992.
11. C.-S. Chang, *Performance Guarantees in Communication Networks.* Telecommunication Networks and Computer Systems, Springer-Verlag, 2000.

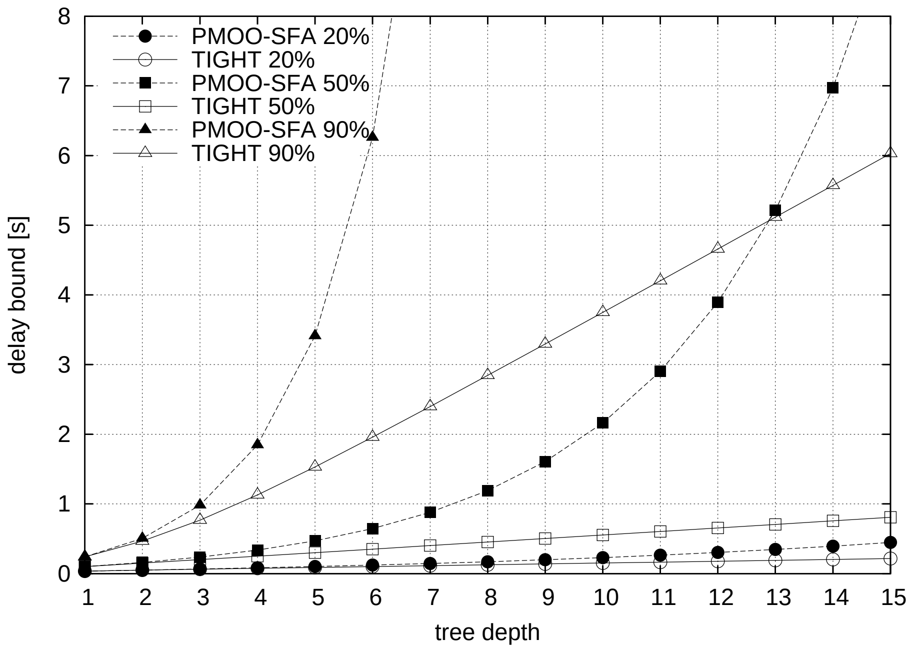

Fig. 7. PMOO-SFA vs. tight bound at several utilizations.

12. 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.
13. 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.
14. 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.
15. 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.
16. H. Kim and J. Hou, "Network calculus based simulation: theorems, implementation, and evaluation," in *Proc. IEEE INFOCOM*, Mar. 2004.
17. M. Fidler, "An end-to-end probabilistic network calculus with moment generating functions," in *Proc. of IEEE IWQoS*, pp. 261–270, Jun 2006.
18. Y. Jiang, "A basic stochastic network calculus," in *Proc. ACM SIGCOMM*, pp. 123–134, Sept. 2006.
19. J. Schmitt and F. Zdarsky, "The DISCO Network Calculator - a toolbox for worst case analysis," in *Proc. of VALUETOOLS*, ACM, Nov. 2006.
20. A. Bouillard and E. Thierry, "An algorithmic toolbox for network calculus," Tech. Rep. RR-6094, Unité de recherche INRIA Rennes, 2007.
21. C.-S. Chang, "Stability, queue length and delay of deterministic and stochastic queueing networks," *IEEE Transactions on Automatic Control*, vol. 39, pp. 913–931, May 1994.
22. M. Andrews, "Instability of fifo in session-oriented networks," in *Proc. SODA*, pp. 440–447, Mar. 2000.
23. A. Charny and J.-Y. L. Boudec, "Delay bounds in a network with aggregate scheduling," in *Proc. QofIS*, pp. 1–13, Sept. 2000.
24. Y. Jiang, "Delay bounds for a network of guaranteed rate servers with fifo aggregation," *Computer Networks*, vol. 40, no. 6, pp. 683–694, 2002.
25. Z.-L. Zhang, Z. Duan, and Y. T. Hou, "Fundamental trade-offs in aggregate packet scheduling," in *Proc. ICNP*, pp. 129–137, Nov. 2001.
26. D. Starobinski, M. Karpovsky, and L. A. Zakrevski, "Application of network calculus to general topologies using turn-prohibition," *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, pp. 411–421, 2003.
27. M. Fidler and V. Sander, "A parameter based admission control for differentiated services networks," *Computer Networks*, vol. 44, no. 4, pp. 463–479, 2004.
28. L. Lenzini, L. Martorini, E. Mingozzi, and G. Stea, "Tight end-to-end per-flow delay bounds in fifo multiplexing sink-tree networks," *Performance Evaluation*, vol. 63, no. 9, pp. 956–987, 2006.

29. G. Urvoy-Keller, G. Hébuterne, and Y. Dallery, “Traffic engineering in a multipoint-to-point network,” *IEEE Journal on Selected Areas in Communications*, vol. 20, pp. 834–849, May 2002.
30. J.-Y. Le Boudec and A. Charny, “Packet scale rate guarantee for non-fifo nodes,” in *Proc. IEEE INFOCOM*, pp. 23–26, June 2002.
31. J.-Y. Le Boudec and G. Rizzo, “Pay bursts only once does not hold for non-fifo guaranteed rate nodes,” *Performance Evaluation*, vol. 62, no. 1-4, pp. 366–381, 2005.
32. J. Echagüe and V. Cholvi, “Worst case burstiness increase due to arbitrary aggregate multiplexing,” in *Proc. of VALUETOOLS*, ACM, Nov. 2006.