A Self-Adversarial Approach to Delay Analysis under Arbitrary Scheduling

Jens B. Schmitt, Hao Wang, and Ivan Martinovic

{jschmitt,wang,martinovic}@cs.uni-kl.de

Abstract. Non-FIFO processing of flows by network nodes is a frequent phenomenon. Unfortunately, the state-of-the-art analytical tool for the computation of performance bounds in packet-switched networks, network calculus, cannot deal well with non-FIFO systems. The problem lies in its conventional service curve definitions. Either the definition is too strict to allow for a concatenation and consequent beneficial end-to-end analysis, or it is too loose and results in infinite delay bounds. Hence, in this paper, we propose a new approach to derive tight bounds in tandems of non-FIFO nodes, the so-called self-adversarial approach. The self-adversarial approach is based on a previously proposed method for calculating performance bounds in feedforward networks [30]. By numerical examples we demonstrate the superiority of the self-adversarial approach over existing methods for the analysis of non-FIFO tandems as well as that for low to medium utilizations it even stays close to corresponding FIFO performance bounds.

1 Introduction

1.1 Motivation

In the recent past, network calculus [10,25] has shown promise as an alternative methodology, besides classical queueing theory, for the performance analysis of packet-switched networks. It has found usage as a basic tool for attacking several important network engineering problems: most prominently in the Internet's Quality of Service proposals IntServ and DiffServ, but also in other environments like wireless sensor networks [22,29], switched Ethernets [31], Systems-on-Chip (SoC) [8], or even to speed-up simulations [21], to name a few. Unfortunately, it comes up short in certain fundamental aspects to really catch on as *the* system theory for the Internet as which it is sometimes advertised (see for the subtitle and the discussion in the introduction of the book by Le Boudec and Thiran [25]). Two prominent fundamental limitations that can be raised are: (1) its deterministic nature, and (2) its dependence on strict FIFO processing of flows. While still being open to some degree, the first issue has been dealt with extensively in literature, and particularly recent approaches towards a stochastic relaxation of network calculus can, for example, be found in [20,11,16]. Yet, the second issue about non-FIFO processing is still largely unexplored. In the next subsection, we provide an overview of the previous scarce work we could find on this topic. In contrast, non-FIFO *multiplexing* between flows was an intensive subject in previous work (see, for example, [30,7] and the references therein to recent contributions on the so-called arbitrary or general multiplexing). In this paper, we are concerned with the scheduling within a flow under analysis, so we deal with the questions *when* work units are processed by a node and *in which order*. The first question on how a node provides its capacity to a flow is flexibly answered by the network calculus concept of service curves, while the second question has so far always been answered by assuming a FIFO processing order. The goal of this paper is to provide a more flexible answer to this questions. In the following, we provide some arguments on why besides being of theoretical interest this issue should be addressed.

Assuming that the work units of a flow under analysis are processed in FIFO order constitutes a logical break for the worst-case methodology in a certain sense, as we discuss now. Assume that a flow traverses a system resulting in a certain output process. The *real delay*[1] for a work unit input at time t and output at time t' is simply defined as

$$rd(t) = t' - t.$$

Any processing order other than FIFO results in an increase of the worst-case real delay; this can be easily seen by the following argument: Assume at time t_0 a work unit which experiences the worst-case real delay is input to a FIFO system. Now assume we can change the processing order of work units. If the work unit is further delayed by scheduling work units that arrived later, then certainly the real delay of that work unit under the new processing order will be worse. Processing that work unit earlier will make its new real delay $rd'(t_0)$ smaller, yet, the work unit which was processed just ahead of the above work unit is now leaving the system when the above work unit would have left, yet that work unit has arrived at time $t_1 \leq t_0$ such that the new real delay of that work unit $rd'(t_1)$ is higher than or equal to the one from the FIFO worst-case work unit, i.e., $rd(t_0) \leq rd'(t_1)$. So, in this sense FIFO can be viewed as the best-case assumption on the processing order of the flow under analysis. Therefore, it can be seen as consistent with a worst-case methodology to release the FIFO processing assumption.

Furthermore, by providing the following real-world examples where non-FIFO behavior is exhibited, we also want to stress the practical relevance of this work:

Packet Reordering: In several studies of Internet traffic it has been shown that packet reordering is a frequent event (see, for example, [4,19]). According to these studies this is occurs because of the growing amount of parallelism on a global (use of multiple paths or parallel links) as well as on a local (device) level. In particular, for scalability reasons high-speed routers often contain a complex multi-stage switching fabric which cannot ensure to preserve the preservation of arrivals at its output. This is due to a common design trade-off where FIFO

[1] *The word* real *is chosen for the purpose of contrasting it to the* virtual *delay, later on defined as delay under FIFO processing of a flow.*

service at the input queues is relaxed in order to avoid head-of-line blocking by choosing from a set of packets in the input queue (often some window-based scheme is used). Furthermore, the use of link aggregation, where multiple physical lines are aggregated into a single virtual link, may often lead to a non-FIFO behavior [6].

Content-Dependent Packet Scheduling: As the last example, let us mention wireless sensor networks (WSN) where packet scheduling decisions may be based on the contents of packets following a WSN-typical data-centric paradigm. Under such circumstances hardly anything can be assumed about the scheduling order, let alone the FIFO behavior.

So, from a methodological as well as an application perspective there is a clear need for an investigation on how network calculus can be extended towards an analysis without any FIFO assumptions. Immediate questions that come up are:

– Can the existing network calculus concepts be carried over to the non-FIFO case?
– Is an efficient end-to-end analysis still possible?
– What is the cost in terms of performance bounds compared to pure FIFO systems?

1.2 Related Work

There is amazingly little existing work on the treatment of non-FIFO systems in the context of network calculus. Remarkably, in his pioneering paper [12], Cruz briefly showed how to derive a delay bound for a single work-conserving server under a general scheduling assumption (comprising any non-FIFO processing order) based on the observation that the maximum backlogged period can be bounded given that traffic is regulated. Similar results can also be found in [10]. Yet, the multiple node case as well as more general server models are not treated therein.

In [24], Le Boudec and Charny investigate a non-FIFO version of the Packet Scale Rate Guarantee ($PSRG$) node model as used in DiffServ's Expedited Forwarding definition. They show that the delay bound from the FIFO case still applies in the single node case while it does not in a specific two node case. They leave more general concatenation scenarios for further study.

In [30], we dealt with the problem of computing tight delay bounds for a network of arbitrary (non-FIFO) *aggregate multiplexers*. They show the tightness of their bounding method by sample path arguments. Yet, in contrast to the problem setting in this paper, we still make a FIFO assumption on the processing order within a flow and only allow for non-FIFO behavior between flows (see the discussion in the previous subsection). Bouillard et al. recently provided more advanced and general results for the same setting in [7], yet nevertheless, they were still based on FIFO processing per flow.

To the best of our knowledge, the only previous work that also tries to derive end-to-end delay bounds without any FIFO assumptions was done by Rizzo and Le Boudec [27]. They investigate delay bounds for a special server model, non-FIFO guaranteed rate (GR) nodes, and show that a previously derived delay bound for GR nodes [17] is not valid for a non-FIFO case (against common belief). Furthermore, they derive a new delay bound based on the network calculus results. Their delay bound no longer exhibits the nice pay-bursts-only-once phenomenon. Based on sample path arguments, they argue that their bound is tight and thus conclude that "pay bursts only once does not hold for non-FIFO guaranteed rate nodes". In contrast, we show that non-FIFO systems may still possess a concatenation property. This seeming contradiction is discussed in more detail at the very end of this paper.

1.3 Contributions

In this work, the following contributions are made:

– We demonstrate difficulties with existing service curve definitions under non-FIFO processing.
– We introduce a new approach, called self-adversarial, that enables a true end-to-end analysis for non-FIFO systems.
– We show that, somewhat contrary to the results presented in literature, the pay-bursts-only-once phenomenon still holds for non-FIFO systems.

2 Preliminaries on Network Calculus

Network calculus is a min-plus system theory for deterministic queueing systems that builds upon the calculus for network delay in [12], [13]. The important concept of service curve was introduced in [2,9,14,23,28]. 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 [11] and also referred to as pay-bursts-only-once phenomenon in [25]. A detailed treatment of min-plus algebra and of network calculus can be found in [3] and [10], [25], respectively.

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

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

In particular, the input function $F(t)$ and the output function $F'(t)$, which cumulatively count for the number of work units that are input to, respectively output from, a system $\mathcal{S}$, are in $\mathcal{F}$. Throughout the paper, we assume in- and output functions to be continuous in both time and space. Note that this is not a general limitation as there exist transformations between discrete and continuous models [25].

There are two important min-plus algebraic operators:

Definition 1. *(Min-plus Convolution and Deconvolution) The min-plus convolution and deconvolution of two functions $f, g \in \mathcal{F}$ are defined to be*

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

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

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 [25]. 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 [25], [10]. Let us now turn to the performance characteristics of flows that can be bounded by network calculus means:

Definition 2. *(Backlog and Virtual 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*

$$b(t) = F(t) - F'(t).$$

The virtual delay *for a work unit input at time t is defined as*

$$vd(t) = \inf\{\tau \ge 0 : F(t) \le F'(t+\tau)\}.$$

So, this is the point where the FIFO assumption sneaks in the network calculus as far as delay is concerned, because $rd(t) = vd(t)$ for all t only under FIFO processing of the flow. We use the usual network calculus terminology of the so-called *virtual* delay in contrast to the *real* delay, as defined above (see Section 1.1). Next, 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*

$$\forall t, s \ge 0, s \le t : F(t) - F(t-s) \le \alpha(s) \Leftrightarrow F = F \otimes \alpha.$$

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

Definition 4. *(Service Curve – SC) 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*

$$F' \ge F \otimes \beta.$$

For continuous functions F and β this is equivalent to the following condition

$$\forall t \ge 0 : \exists s \le t : F'(t) \ge F(s) + \beta(t-s).$$

A typical example of a service curve is given by a so-called rate-latency function $\beta_{R,T}(t) = R(t-T)\cdot 1_{\{t>T\}}$, where $1_{\{cond\}}$ is 1 if the condition *cond* is satisfied and 0 otherwise. Also, nodes operating under a delay-based scheduler and guaranteeing that a work unit arriving at any time t will leave the node at time $t' \le t+T$ for some fixed $T>0$, i.e. $\forall t \ge 0 : rd(t) \le T$, are known to provide a service curve $\delta_T = \infty \cdot 1_{\{t>T\}}$. We also call these *bounded latency* nodes.

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:*

$$\textit{backlog: } \forall t : b(t) \le (\alpha \oslash \beta)(0) =: v(\alpha,\beta),$$

$$\textit{virtual delay: } \forall t : vd(t) \le \sup_{t\ge 0} \inf\{\tau \ge 0 : \alpha(t) \le \beta(t+\tau)\} =: h(\alpha,\beta),$$

$$\textit{output (arrival curve } \alpha' \textit{for } F'\textit{): } \alpha' = \alpha \oslash \beta.$$

Here, note again that the delay bound is only a *virtual* one, meaning that it is based on the FIFO assumption for the flow under analysis. One of the strongest results of the network calculus 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 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 achieves tight performance bounds, which in general is not the case for an iterative per-node application of Theorem 1.

3 Conventional Network Calculus And Non-FIFO Systems

In this section, we investigate how the existing network calculus can cope with non-FIFO systems. The crucial aspect is the node model. We start with the typical service curve model as defined in the previous section and then turn to strict service curves, only to find out that both of them encounter problems under non-FIFO processing.

3.1 Using Service Curves (SC) for Non-FIFO Systems

As the *SC* definition bears the advantages that many systems belong to that class and that it possesses a concatenation property, it is worthwhile an attempt to apply it also in the case of non-FIFO systems. Yet, the following example shows that it is impossible to bound the real delay in non-FIFO systems solely based on the *SC* definition:

Example 1. (*SC* Cannot Bound the Real Delay) Assume a single node system $\mathcal{S}$ which offers a rate-latency service curve $\beta = \beta_{2,1}$ to a flow F which is constrained by an affine arrival curve $\alpha = \gamma_{1,0}$. Now assume the flow to be greedy, that means $F = \alpha$ and the server to be lazy, that means $F' = F \otimes \beta$. Thus, we obtain

$$\begin{aligned} F' &= \alpha \otimes \beta = \gamma_{1,0} \otimes \beta_{2,1} = \gamma_{1,0} \otimes \gamma_{2,0} \otimes \delta_1 \\ &= (\gamma_{1,0} \wedge \gamma_{2,0}) \otimes \delta_1 \leq \gamma_{1,0} \otimes \delta_1 < \gamma_{1,0} = F. \end{aligned}$$

Hence, $\forall t \geq 0 : F'(t) < F(t)$, or equivalently, $\forall t \geq 0 : b(t) > 0$, which means that the system remains backlogged the entire time. Therefore, without any assumptions on the processing order, a certain work unit can, under these circumstances, be kept forever in the system. Thus, the real delay of that work unit is unbounded. Note that using the standard FIFO processing assumption, we can of course bound the real delay of the system by $\forall t \geq 0 : rd(t) = vd(t) \leq \frac{3}{2}$.

From this example, we see that the *SC* property is too weak as a node model for analyzing non-FIFO systems. Therefore, it is sensible to look for more stringent node models, as it is done in the following subsection.

3.2 Using Strict Service Curves (S^{2}C) for Non-FIFO Systems

A number of systems provides more stringent service guarantees than captured by *SC*, fulfilling the so-called strict service curve [25] (also known as strong service curve [15,2] and related to the universal service curve concept in [26])

Definition 5. *(Strict Service Curve – S^2C) 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 $\beta(u)$. A backlogged period of duration u at time t is defined by the fact that $\forall s \in (t-u, t] : b(s) > 0$.*

Note that any node satisfying S^2C also satisfies *SC*, but not vice versa. For example, a bounded latency node does not provide δ_T as a *strict* service curve. In fact, it does not provide any S^2C apart from the trivial case $\beta = 0$. On the other hand, there are many schedulers that offer strict service curves; for example, most of the generalized processor sharing-emulating schedulers (e.g., PGPS [26], WF^{2}Q [5], or round robin schedulers like SRR [18], to name a few), offer a strict service curve of the rate-latency type.

Now for bounding the real delay under S^2C: In fact, as was already shown by Cruz [12] (and can also be found in [10] (Lemma 1.3.2)), the intersection point between an arrival and a *strict* service curve constitutes a bound on the length of the maximum backlogged period and thus also a bound on the real delay for such a system:

Theorem 3. *(Real Delay Bound for Single S^2C Node) Consider a system $\mathcal{S}$ that offers a strict service curve β. Assume a flow F traversing the system has an arrival curve α. Then we obtain the following bound on the real delay:*

$$rd(t) \leq \sup\{s \geq 0 : \alpha(s) \geq \beta(s)\} =: i(\alpha, \beta).$$

So, the situation has improved in comparison to the SC case: Based on the single node result one can conceive, for the multiple node case, an iterative application of Theorem 3 together with the output bound from Theorem 1. More specifically, if a tandem of n S^2C non-FIFO nodes, each providing a strict service curve $\beta_j, j = 1, \ldots, n$, is to be traversed by an α-constrained flow then a bound on the real delay can be calculated as

$$rd(t) \leq \sum_{j=1}^{n} i(\alpha \oslash \bigotimes_{k=1}^{j-1} \beta_k, \beta_j).$$

Setting for example $\beta_j = \beta_{R,T}, j = 1, \ldots, n$ and $\alpha = \gamma_{r,b}$ this results in

$$rd(t) \leq \frac{n(b + RT) + \frac{n}{2}(n-1)rT}{R - r}. \tag{1}$$

Here, we see a typical drawback of additive bounding methods, with the burst of the traffic being paid n times as well as a quadratic scaling of the bound in the number of nodes [11,25]. The key to avoid this behavior is to perform an end-to-end analysis based on a concatenation theorem. Yet, as known and demonstrated in the next example, S^2C does not possess such a concatenation property.

Example 2. (S^2C Possesses No Concatenation Property) Assume two systems $\mathcal{S}_1$ and $\mathcal{S}_2$, both providing a strict rate-latency service curve $\beta^i = \beta_{1,1}, i = 1, 2$, which are traversed in sequence by a flow F. Let F_1' and F_2' be the output functions from $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. As a candidate strict service curve for the composite system, we consider $\beta^{1,2} = \beta^1 \otimes \beta^2 = \beta_{1,2}$.

We now construct a backlogged period $[t_1, t_2]$ of the composite system such that

$$F_2'(t_2) - F_2'(t_1) < \beta^{1,2}(t_2 - t_1).$$

thereby showing that $\beta^{1,2}$ is not a strict service curve for the composite system:

Let $t_1 = 0$ and $t_2 = 3$ and assume the following behavior of the input and output function

$$F(t) = \begin{cases} \epsilon \ 0 < t < 2 \\ 2\epsilon \ 2 \leq t \leq 3 \end{cases}, \qquad F_1'(t) = \begin{cases} 0 \ 0 \leq t \leq 1 \\ \epsilon \ 1 < t \leq 3 \end{cases},$$

$$F_2'(t) = \begin{cases} 0 \ 0 \leq t \leq 2 \\ \epsilon \ 2 < t \leq 3 \end{cases},$$

with any $\epsilon > 0$. It is easy to check that the composite system is continuously backlogged during $[0, 3]$ as well as that each individual system is not violating its strict service curve property. Nevertheless, for any choice of $\epsilon < 1$ we obtain

$$F_2'(3) - F_2'(0) = \epsilon < \beta^{1,2}(3) = 1,$$

which shows that $\beta^{1,2}$ is not S^2C for the composite system (while, of course, being SC for it). In fact, by extending the example appropriately it can be shown that the only strict service curve that can be guaranteed by the composite system is the trivial case $\beta = 0$. This can be seen by making ϵ arbitrarily small and alternating between backlogged and idle periods of the individual systems sufficiently often. Another way to view this, is that the backlogged period of a composite system cannot be bounded based on the individual systems providing a strict service curve.

4 The Self-Adversarial Approach

In this section, we devise an approach, called the self-adversarial method, to compute a tight delay bound for non-FIFO systems based on a technique that was introduced in [30].

4.1 The Self-Adversarial Method

As briefly discussed in Section 1.2, in [30], we proposed a technique for computing tight delay bounds in the network of arbitrary (non-FIFO) *aggregate multiplexers*, yet we still made a FIFO processing order assumption per flow. So, this technique is not directly applicable when releasing all FIFO assumptions and besides arbitrary multiplexing also assumes arbitrary scheduling within a flow. Nevertheless, there is a way to exploit the proposed method for the problem at hand by transforming the arbitrary scheduling problem into an arbitrary aggregate multiplexing problem. More specifically, we split the original flow, with the arrival curve α, into two sub-flows: one with the arrival curve $\alpha_1 = \gamma_{0,\epsilon}$ and the other one with the arrival curve $\alpha_2 = \alpha - \gamma_{0,\epsilon}$. Both flows traverse the same servers as the original flow. This transformation is illustrated in Figure 1.

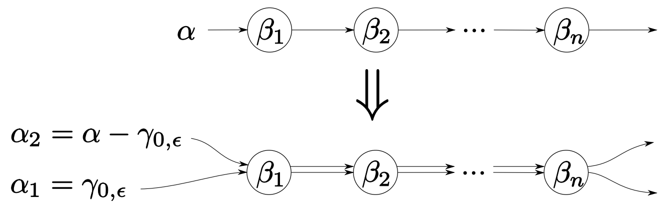

α
β_1
β_2
$\cdots$
β_n
$\alpha_2 = \alpha - \gamma_{0,\epsilon}$
$\alpha_1 = \gamma_{0,\epsilon}$

Fig. 1. Transformation of the pure non-FIFO problem into an arbitrary aggregate multiplexing problem.

Now the method from [30] allows us to find the maximum left-over end-to-end service curve under arbitrary multiplexing, i.e., under any possible interleaving of the two sub-flows. To that end, the problem is reformulated as an optimization problem that can be solved by using standard methods. In [30,7], it is shown that this approach achieves tight delay bounds. So, in our case we can proceed with the following steps:

1. Computation of the left-over service curve for sub-flow 1 according to [30]: $\beta_1^{l.o.}$.
2. Computation of the delay bound for sub-flow 1: $d_1(\epsilon) = h\left(\alpha_1, \beta_1^{l.o.}\right)$.
3. Letting the delay bound for sub-flow 1 go to the limit: $d = \lim_{\epsilon \to 0} d_1(\epsilon)$.

What is effectively done here, is to assume that a part of the flow pretends to be an adversary to the other part of the flow when it comes to competition for the forwarding resources. This is why we call it the *self-adversarial* method. Taking this behavior to the limit, i.e., making the adversary part as large as possible, gives us a real delay bound as experienced by a single (infinitesimally small) work unit.

We remark that the computation of the horizontal deviation in step 2 implicitly makes a FIFO assumption for sub-flow 1. Yet, in the limit this is not a problem because a single work unit provides no degrees of freedom for the processing order any more.

Note that for the splitting of the original flow into two sub-flows we assumed that $\epsilon > 0$ is chosen such that $\alpha_2 \geq 0$. In fact, for some arrival curves this may not be possible. More precisely, if $\alpha(t)$ is continuous at $t = 0$ (e.g., a constant rate arrival curve), then the splitting described is not feasible. In such cases, the original arrival curve should be shifted to the left by some small amount Δ and set to zero for $t \leq 0$. The approach is then performed on this new (strictly larger) arrival curve. To find the delay bound under the original arrival curve, one lets $\Delta \to 0$. We decided to neglect this (rarely occurring) technicality in the above description of the self-adversarial method in order not to (further) complicate it.

4.2 Self-Adversarial vs. Additive Bounding Method

Let us investigate by a simple example how the self-adversarial method works and also compare it to an additive bounding based on S^2C. Assume a token-bucket arrival curve $\gamma_{r,b}$ for the flow under investigation ($b > 0$), which traverses two servers providing *strict* rate-latency service curves $\beta_{R_iT_i}, i = 1, 2$. According to the additive bounding based on S^2C the delay bound then becomes:

$$d^{AD} = T_1 + T_2 + \frac{b + rT_1}{R_1 - r} + \frac{b + r\left(T_1 + T_2\right)}{R_2 - r}.$$

For the self-adversarial method we first split the flow into two sub-flows: subflow 1 with $\gamma_{0,\epsilon}$ and subflow 2 with $\gamma_{r,b-\epsilon}$ as arrival curves. Proceeding with the steps described in the previous section we obtain the following delay bound:

1. Computation of the left-over service for sub-flow 1 according to [30]:
$$\beta_1^{l.o.} = \beta_{\min\{R_1,R_2\}-r, T_1+T_2+\frac{b-\epsilon+rT_1}{\min\{R_1,R_2\}-r}+\frac{rT_2}{R_2-r}}.$$
2. Computation of the delay bound for sub-flow 1:
$$d_1(\epsilon) = \frac{\epsilon}{\min\{R_1, R_2\} - r} + T_1 + T_2 + \frac{b - \epsilon + rT_1}{\min\{R_1, R_2\} - r} + \frac{rT_2}{R_2 - r}.$$

3. Letting the delay bound for sub-flow 1 go to the limit ($\epsilon \to 0$):

$$d^{SA} = T_1 + T_2 + \frac{b + rT_1}{\min\{R_1, R_2\} - r} + \frac{rT_2}{R_2 - r}.$$

A simple inspection shows that $d^{SA} \leq d^{AD}$, where equality only holds if $b = 0 \wedge (T_1 = 0 \vee r = 0)$, which are strong restrictions. Hence, this demonstrates that the additive method is not tight under most circumstances. Similar problems with purely min-plus algebraic methods are reported and extensively discussed in [30]. These problems are inherent in using the min-plus algebraic approach. In particular, by the application of a min-plus convolution the knowledge on the *order of servers* is lost. Yet, this order is crucial to derive tight delay bounds for non-FIFO systems. The min-plus algebraic approach automatically maps a tandem of system to the worst-case order it could be in (see [30] for more discussions along this line).

So, with respect to the tightness of the computed bounds, the self-adversarial method is superior to the additive method. A potential drawback for the self-adversarial method is that the computational effort for the self-adversarial method can become very high. In particular, if arrival and service curves are piecewise-linear functions then a set of optimization problem needs to be solved first before the final left-over service curve can be constructed according to [30] (in [7] a more efficient and provably tight approach is proposed, on which the self-adversarial method could also be based). The cardinality of that set grows exponentially in the number of nodes traversed and may quickly become prohibitive. For details see [30], or even better [7], which also demonstrates the problem of computing a tight delay bound under arbitrary multiplexing in general feedforward networks to be NP-hard.

We also remark that the self-adversarial method requires S^2C servers (as in other cases like, e.g., fixed priority schedulers or arbitrary multiplexing scenarios). This requirement is crucial for setting up the optimization problem in [30] and a relaxation towards only assuming SC seems infeasible. This means, in particular, that bounded latency nodes cannot be analyzed. Similarly, according to [30], the self-adversarial method can only be applied to piecewise-linear concave arrival and convex service curves. Such a restriction does not apply, in principle, to the additive bounding method.

While the tightness of the self-adversarial method is "inherited" from [30], it can also be understood in the original system. In particular, if the processing order applied is to always choose the work unit that has entered the *network* last (assuming work units are time-stamped when they enter the network) then we conjecture that the bound can actually be achieved. This processing order has also been coined shortest-in-system (SIS) in the realm of adversarial queueing theory [1]. If only one node is traversed, then SIS becomes LIFO and clearly constitutes the worst-case processing order. In multi-node scenarios, we conjecture that SIS produces a worst-case sample path if greedy arrivals (exactly following the arrival curve) and lazy servers (exactly following the service curve) are assumed.

As the last remark, we note that if there is also cross-traffic from other flows we can first apply [30] to derive a left-over service curve for the flow of interest and then apply the self-adversarial method to arrive at tight bounds under arbitrary multiplexing *and* scheduling, i.e., a completely non-FIFO scenario.

5 Numerical Experiments

To give some feeling for the improvements achievable by using the self-adversarial approach compared to an additive bounding based on S^2C we provide some numerical experiments. In addition, we demonstrate what cost is incurred when releasing the FIFO assumption. For these numerical experiments we use simple settings: as arrival curve for the flow to be analyzed we assume a token bucket $\gamma_{r,b}$ where we set $r = 10[Mbps]$ and $b = 5[Mb]$ (unless we vary the rate r to achieve a certain utilization); for the service curves of the nodes to be traversed we use a rate-latency function $\beta_{R,T}$ with $R = 20[Mbps]$ and $T = 0.01[s]$. Unless we use the number of nodes as a primary factor in the experiments we assume $n = 10$ nodes to be traversed by the flow under investigation.

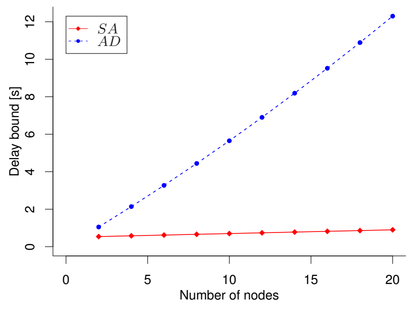
5.1 Comparison of Self-Adversarial and Additive Bounding

In this first set of numerical experiments we investigate how the self-adversarial (SA) and additive (AD) bounding methods compare to each other. In Figure 2(a) the two methods are shown for a varying number of nodes (from 2 to 20). To emphasize the bad scaling of the additive method we also provide results for the same experiment with a larger number of nodes to be traversed (up to 100) in Figure 2(b). In both graphs it is obvious that the end-to-end analysis facilitated by the self-adversarial approach is highly superior and scales linearly with the number of nodes, whereas the additive bounding method scales quadratically with the number of nodes traversed and thus becomes a very conservative bound quickly.

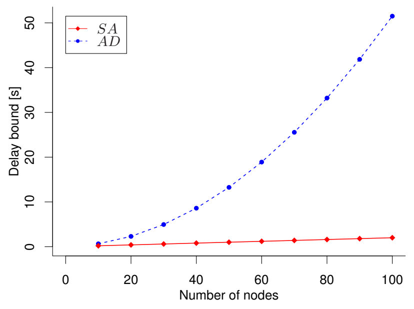
A different view on the relative performance of self-adversarial and additive methods is provided in Figure 2(c). Here, the acceptable utilizations (captured by the ratio of the the rate for the flow under investigation and the service rate of the tandem, i.e., $\frac{r}{R}$) for a given delay bound are shown for both methods. This information can be used for admission control purposes. Again, as can be clearly seen, the self-adversarial method outperforms the additive bounding by far, especially for lower delay bounds. For example, if we desire a delay bound of 2s, then an admission control using the additive bounding would return with an infeasible reply, whereas under the self-adversarial approach we could admit traffic up to $\approx 80\%$ of the service rate.

5.2 FIFO vs. Non-FIFO Delay Bounds

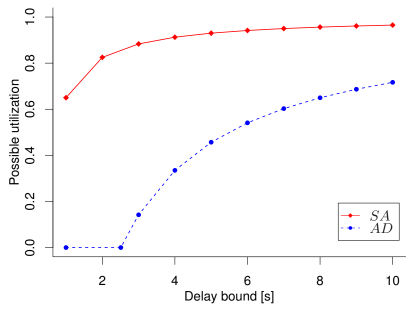
In the next set of numerical experiments, we investigate the cost of releasing the FIFO assumption in terms of delay bounds. For that purpose, we vary the



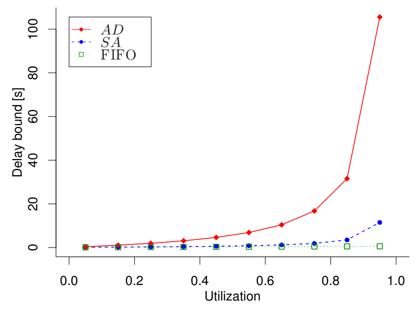
(a) Delay bounds for self-adversarial and additive bounding methods.



(b) Exposing the quadratic scaling of the additive bound.



(c) Possible utilizations for a target delay bound under *SA* and *AD*.



(d) FIFO vs. non-FIFO delay bounds depending on the utilization.

Fig. 2. Comparison of self-adversarial approach to other analysis methods under different metrics. Subfigures (a) and (b) show results for 50% utilization.

utilization by increasing the sustained rate of the traffic flow under investigation (while at the same time scaling the bucket depth proportionally). As we can observe from Figure 2(d), only for higher utilizations there is a significant difference between the FIFO and non-FIFO delay bounds if using the self-adversarial bounding approach. On the other hand, if the additive bounding was used, the cost of releasing FIFO assumptions is high, which may be why FIFO behavior is often assumed a necessary condition to achieve good delay bounds [27]. Yet, under strict service curve assumptions and using the self-adversarial approach this assumption is not necessarily required any more.

From an application perspective, the bottom line is that only for highly utilized systems it is necessary to enforce a FIFO behavior, as far as delay bounds are concerned. For systems with lower utilizations, optimizations such as for example link aggregation or multi-stage switching fabrics do not incur a high cost in terms of worst-case delay bounds.

6 Conclusion and Discussion

In this paper, it was our goal to extend the scope of network calculus towards non-FIFO systems, as non-FIFO behavior is a reality in many networking scenarios. It turned out that the existing service curve definitions are not satisfying under non-FIFO scheduling: they are either too loose to enable any bounding or too strict to allow for an efficient end-to-end analysis. Therefore, we devised a new approach, called the self-adversarial bounding method, which is based on previous work of ours and is provably tight. By numerical examples, we showed that the self-adversarial approach is far superior to existing methods.

The self-adversarial approach allows to recover the pay-bursts-only-once phenomenon for non-FIFO systems, which had been disputed to be valid under non-FIFO scheduling in literature [27]. This seeming contradiction is due to different assumptions on the service provided by nodes, guaranteed rate as in [27], or strict service curve, as in this paper. Since the concatenation of guaranteed rate nodes is based on their equivalence to rate-latency service curves (modulo packetization effects), a convolution of them only provides an SC guarantee and thus cannot bound the real delay, as discussed in Section 3. Hence, the only resort is an additive bounding which, however, cannot recover the pay-bursts-only-once phenomenon for the arbitrary scheduling case.

References

1. P. Raghavan M. Sudan A. Borodin, J. M. Kleinberg and D. P. Williamson. Adversarial queuing theory. *Journal of the ACM*, 48(1), 2001.
2. R. Agrawal, R. L. Cruz, C. Okino, and R. Rajan. Performance bounds for flow control protocols. *IEEE/ACM Transactions on Networking*, 7(3):310–323, June 1999.
3. 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.
4. J. C. R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Trans. Netw.*, 7(6):789–798, 1999.
5. J. C. R. Bennett and H. Zhang. WF^{2}Q: Worst-case fair weighted fair queueing. In *Proc. IEEE INFOCOM*, pages 120–128, March 1996.
6. J. M. Blanquer and B. Özden. Fair queuing for aggregated multiple links. *SIGCOMM Comput. Commun. Rev.*, 31(4):189–197, 2001.
7. A. Bouillard, L. Jouhet, and E. Thierry. Tight performance bounds in the worst-case analysis of feed-forward networks. In *Proc. IEEE INFOCOM*, pages 1–9, March 2010.
8. S. Chakraborty, S. Kuenzli, L. Thiele, A. Herkersdorf, and P. Sagmeister. Performance evaluation of network processor architectures: Combining simulation with analytical estimation. *Computer Networks*, 42(5):641–665, 2003.
9. C.-S. Chang. On deterministic traffic regulation and service guarantees: A systematic approach by filtering. *IEEE Transactions on Information Theory*, 44(3):1097–1110, May 1998.
10. C.-S. Chang. *Performance Guarantees in Communication Networks.* Telecommunication Networks and Computer Systems. Springer-Verlag, 2000.

11. F. Ciucu, A. Burchard, and J. Liebeherr. A network service curve approach for the stochastic analysis of networks. In *Proc. ACM SIGMETRICS*, pages 279–290, June 2005.
12. R. L. Cruz. A calculus for network delay, Part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, January 1991.
13. R. L. Cruz. A calculus for network delay, Part II: Network analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, January 1991.
14. R. L. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal on Selected Areas in Communications*, 13(6):1048–1056, August 1995.
15. R. L. Cruz and C. M. Okino. Service guarantees for window flow control. In *Proc. 34th Allerton Conf. Communications, Control, and Computing*, October 1996.
16. Markus Fidler. An end-to-end probabilistic network calculus with moment generating functions. In *Proc. of IEEE IWQoS*, pages 261–270, Jun 2006.
17. P. Goyal, S. S. Lam, and H. M. Vin. Determining end-to-end delay bounds in heterogeneous networks. *Multimedia Syst.*, 5(3):157–163, 1997.
18. C. Guo. SRR: An O(1) time complexity packet scheduler for flows in multi-service packet networks. *IEEE/ACM Transactions on Networking*, 12(6):1144–1155, December 2004.
19. S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Measurement and classification of out-of-sequence packets in a tier-1 IP backbone. *IEEE/ACM Trans. Netw.*, 15(1):54–66, 2007.
20. Y. Jiang. A basic stochastic network calculus. In *Proc. ACM SIGCOMM*, pages 123–134, September 2006.
21. H. Kim and J.C. Hou. Network calculus based simulation: theorems, implementation, and evaluation. In *Proc. IEEE INFOCOM*, March 2004.
22. A. Koubaa, M. Alves, and E. Tovar. Modeling and worst-case dimensioning of cluster-tree wireless sensor networks. In *Proc. of RTSS'06)*, pages 412–421, Rio de Janeiro, Brazil, 2006. IEEE Computer Society.
23. J.-Y. Le Boudec. Application of network calculus to guaranteed service networks. *IEEE Transactions on Information Theory*, 44(3):1087–1096, May 1998.
24. J.-Y. Le Boudec and A. Charny. Packet scale rate guarantee for non-fifo nodes. In *Proc. IEEE INFOCOM*, pages 23–26, June 2002.
25. J.-Y. Le Boudec and P. Thiran. *Network Calculus A Theory of Deterministic Queuing Systems for the Internet.* Number 2050 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, 2001.
26. A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, June 1993.
27. G. Rizzo and J.-Y. Le Boudec. Pay bursts only once does not hold for non-fifo guaranteed rate nodes. *Performance Evaluation*, 62(1-4):366–381, 2005.
28. H. Sariowan, R. L. Cruz, and G. C. Polyzos. Scheduling for quality of service guarantees via service curves. In *Proc. IEEE ICCCN*, pages 512–520, September 1995.
29. J. Schmitt and U. Roedig. Sensor network calculus - a framework for worst case analysis. In *Proc. of DCOSS*, pages 141–154, June 2005.
30. J. Schmitt, F. Zdarsky, and M. Fidler. Delay bounds under arbitrary aggregate multiplexing: When network calculus leaves you in the lurch... In *Proc. IEEE INFOCOM*, April 2008.
31. T. Skeie, S. Johannessen, and O. Holmeide. Timeliness of real-time IP communication in switched industrial ethernet networks. *IEEE Transactions on Industrial Informatics*, 2(1):25–39, February 2006.