Calculating Accurate End-to-End Delay Bounds – You Better Know Your Cross-Traffic

Steffen Bondorf
Distributed Computer Systems (DISCO) Lab
University of Kaiserslautern, Germany
bondorf@cs.uni-kl.de

Jens B. Schmitt
Distributed Computer Systems (DISCO) Lab
University of Kaiserslautern, Germany
jschmitt@cs.uni-kl.de

ABSTRACT

Bounds on the end-to-end delay of data flows play a crucial role in different areas, ranging from certification of hard real-time communication capabilities to quality of experience assurance for end users. Deterministic Network Calculus (DNC) allows to derive worst-case delay bounds; for instance, DNC is applied by the avionics industry to formally verify aircraft networks against strict delay requirements. Calculating tight end-to-end delays, however, was proven to be NP-hard. As a result, analyses focus on deriving fairly accurate bounds with feasible effort. Previous work constantly improved on capturing flow scheduling and cross-traffic multiplexing effects on the analyzed flow's path. In contrast, we present an enhanced analysis of the cross-traffic itself to decrease the bound on its worst-case data arrivals that interfere with the analyzed flow. This improvement is beneficial for both of effects, scheduling and multiplexing. By replacing the currently used procedure to bound cross-traffic arrivals with our new method, we can improve network calculus accuracy considerably – we demonstrate improvements that reduce the worst-case delay bound by more than factor 6.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems

Keywords

Cross-traffic arrivals, delay bounds, network calculus

1. INTRODUCTION

Deterministic network calculus (DNC) is increasingly used to analyze networks designed from scratch for delay sensitive employment. Examples can especially be found in the avionics domain where the analysis of AFDX (Avionics Full-Duplex Switched Ethernet) networks has continuously seen attention, e.g., [10, 6, 11]; current Airbus aircraft's AFDX backbone was certified using the algebraic techniques provided by DNC. Calculating a flow's *tight* end-to-end delay in larger networks, however, requires great effort to capture the worst-case flow scheduling and cross-traffic multiplexing throughout the entire network. Therefore, the majority of network calculus literature provides algebraic analyses to derive *upper bounds* on a flow's end-to-end delay. These analyses proceed in two steps:

1. First, the analysis abstracts from the feed-forward network to the analyzed flow's tandem of servers. This step is enabled by bounding the arrivals of cross-traffic at the locations of interference. The next step need not consider the part of the network traversed by cross-flows nor the interference pattern they are subject to.

2. A tandem analysis on the analyzed flow's path constitutes the second step. The flow's end-to-end service curve is derived and the delay bound computed.

So far, DNC analysis following this procedure have seen improvements in order to achieve more accurate bounds. They mainly focussed on capturing the effects on the analyzed flow's path in the left-over service curve closely [13, 18, 17].

For networks of arbitrary multiplexing servers, i.e., any potential reordering of flows is captured, the work presented in [5] constitutes an exception to this overall approach: It transforms the network calculus description of the entire feed-forward network into a set of linear programs (LPs) to optimize – each LP describes one potential scenario of interference between flows. Optimizing all LPs results in the tight delay bound, however, the amount of linear programs and thus the computational complexity grows possibly (super-) exponentially with the network size. The authors prove that their approach to obtain tight bounds is in fact NP-hard.

In this paper, we continue to contribute to the accuracy of algebraically derived network calculus delay bounds. This approach has proven computationally feasible even for large network sizes [3] and open-source tool support is available [2]. In contrast to previous work, however, we focus on the first step and derive a method for better bounds on the arrival of cross-traffic – a property known to be of vital significance in embedded real-time communication systems in the vehicular domain [20]. Thus, our approach to tighten the end-to-end delay bounds in feed-forward networks complements the existing improvements located in the second step of the common algebraic analysis procedure.

Outline. The paper is structured as follows: In Section 2 we provide the background on algebraic network calculus –

VALUETOOLS 2015, December 14-16, Berlin, Germany

DOI 10.4108/eai.14-12-2015.2262565

from the system description establishing worst-case semantics to the analyses bounding the delay a flow suffers when crossing a feed-forward network. In Section 3, we contribute our new cross-traffic arrival bounding method and prove its superiority over the existing procedure. Section 4 evaluates the impact of our improvement on the delay bounds in different Erdős-Rényi networks as well as an AFDX topology and Section 5 concludes the paper.

2. NETWORK CALCULUS BACKGROUND

2.1 The System Description

Network calculus was built around a simple system description [9] consisting of two parts:

Data Arrivals and Forwarding Service

Flows are characterized by functions cumulatively counting their data. They belong to the set $\mathcal{F}_0$ of non-negative, wide-sense increasing functions:

$$\mathcal{F}_0 = \left\{ f : \mathbb{R} \to \mathbb{R}^+_\infty \mid f(0) = 0,\ \forall s \le t\ :\ f(s) \le f(t) \right\},$$
$$\mathbb{R}^+_\infty \coloneqq [0, +\infty) \cup \{+\infty\}\ .$$

We are particularly interested in the functions $A(t)$ and $A'(t)$ cumulatively counting a flow's data put into a server s and put out from s, both up until time t. These functions allow for a straight-forward derivation of flow delays.

Definition 1. (Flow Delay) *Assume a flow with input A crosses a server s and results in the output A'. The (virtual) delay for a data unit arriving at time t is*

$$D(t) = \inf \left\{ \tau \ge 0 \mid A(t) \le A'(t+\tau) \right\}.$$

Note that the order of data within the flow needs to be retained for the (virtual) delay calculation [16].

Network calculus operates in the interval time domain, i.e., its functions of $\mathcal{F}_0$ bound the maximum data arrivals of a flow during any duration of length d.

Definition 2. (Arrival Curve) *Given a flow with input A, a function $\alpha \in \mathcal{F}_0$ is an arrival curve for A iff*

$$\forall t\, \forall d\ 0 \le d \le t\ :\ A(t) - A(t-d) \le \alpha(d).$$

AFDX networks reserve resources for a maximum packet size b periodically sent with a minimum inter-arrival time t_δ such that flows have a maximum data arrival rate of $r = \frac{b}{t_\delta}$ in the fluid model of $\mathcal{F}_0$. This shape of arrival curve is commonly referred to as token bucket and belongs to the class $\mathcal{F}_{\mathrm{TB}} \subseteq \mathcal{F}_0$:

$$\mathcal{F}_{\mathrm{TB}} = \{ \gamma_{r,b} \mid \gamma_{r,b}(0) = 0,\ \forall d > 0\ :\ \gamma_{r,b}(d) = b + r \cdot d \}.$$

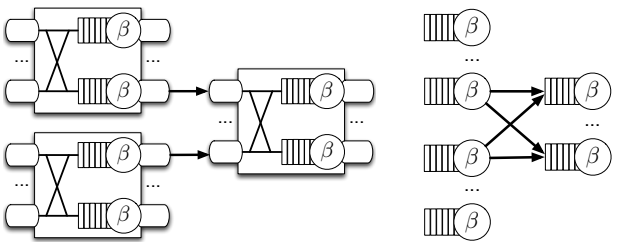
Scheduling and buffering leading to the output function $A'(t)$ depend on a server's forwarding service. It is lower bounded in interval time as well.

Definition 3. (Service Curve) *If the service provided by a server s for a given input A results in an output A', then s offers a service curve $\beta \in \mathcal{F}_0$ iff*

$$\forall t\ :\ A'(t) \ge \inf_{0 \le d \le t} \{ A(t-d) + \beta(d) \}.$$

For instance, service offered by ethernet connections can be described by rate-latency curves $\mathcal{F}_{\mathrm{RL}} \subseteq \mathcal{F}_0$:

$$\mathcal{F}_{\mathrm{RL}} = \{ \beta_{R,T} \mid \beta_{R,T}(d) = \max\{0, R \cdot (d - T)\} \}.$$



(a) Network Graph. (b) Server Graph.

Figure 1: A graph of network devices with output buffering (a) and its server graph connecting the devices' queues (b).

A number of servers fulfill a stricter definition of service curves that guarantees a higher output during periods of queued data, the so-called backlogged periods of a server.

Definition 4. (Strict Service Curve) *Let $\beta \in \mathcal{F}_0$. Server s offers a strict service curve β to a flow iff, during any backlogged period of duration d, the output of the flow is at least equal to $\beta(d)$.*

The Network

In general, networks are modeled as graphs where a node represents a network device like a router or a switch. Devices can have multiple inputs and multiple outputs to connect to other devices (Figure 1a). This network model does not fit well with network calculus' server model for queueing analysis. DNC therefore analyzes so-called *server graphs*. Assuming that a network device's input buffer is served at line speed, queueing effects manifest at the output buffers. These are modeled by the graph's servers (see Figure 1b). AFDX equipment employs output buffering [8]. In wireless sensor networks, nodes usually possess a single transmitter. Thus, one sensor node corresponds to one server and the transmission range defines the server graph's links [1, 3].

2.2 Algebraic Network Calculus

Network calculus was cast in a $(\min, +)$-algebraic framework in [13, 7]. We will first depict the basic operations and then present their combination for flow analysis.

(min,+)-Operations

The following operations allow to manipulate arrival and service curve while retaining their worst-case semantic.

Definition 5. ((min,+)-Operations) *The (min,+)-aggregation, -convolution and -deconvolution of two functions $f, g \in \mathcal{F}_0$ are defined as*

$$\begin{aligned}
\text{aggregation: } (f+g)(t) &= f(t) + g(t), \\
\text{convolution: } (f \otimes g)(t) &= \inf_{0 \le s \le t} \{ f(t-s) + g(s) \}, \\
\text{deconvolution: } (f \oslash g)(t) &= \sup_{u \ge 0} \{ f(t+u) - g(u) \}.
\end{aligned}$$

The system description's service curve definition then translates to $A' \ge A \otimes \beta$, the arrival curve definition to $A \otimes \alpha \ge A$, and performance characteristics can be bounded with the deconvolution $\alpha \oslash \beta$:

Theorem 1. (Performance Bounds) *Consider a server s that offers a service curve β. Assume a flow f with arrival*

Quantifier	Definition
foi	Flow of interest
$\mathbb{F}$	Aggregate of flows
$\{f_n, ..., f_m\}$	Flow aggregate containing flows $f_n, ..., f_m$
$F(s)$	Set of flows at server s
$F_{\text{src}}(s)$	Set of flows originating at server s
$x(f)$, $x(\mathbb{F})$	Cross-traffic of flow f, aggregate $\mathbb{F}$
$\langle s_x, \ldots, s_y \rangle$	Tandem of consecutive servers s_x to s_y
α^f, $\alpha^{\mathbb{F}}$	Arrival curve of flow f, set of flows $\mathbb{F}$
α_s^f, $\alpha_s^{\mathbb{F}}$	Arrival bound at server s
β_s	Service curve of server s
$\beta^{\text{l.o.}f}$, $\beta^{\text{l.o.}\mathbb{F}}$	Left-over service curve

Table 1: Network Calculus Notation.

curve α traverses the server. Then we obtain the following performance bounds for f:

$$\textit{delay: } \forall t \in \mathbb{R}^+ : \; D(t) \leq \inf\{d \geq 0 \,|\, (\alpha \oslash \beta)(-d) \leq 0\},$$
$$\textit{output: } \forall d \in \mathbb{R}^+ : \; \alpha'(d) = (\alpha \oslash \beta)(d),$$

where the delay bound holds independent of t and α' is an arrival curve for A'.

Analyzing an entire flow with cross-traffic on its path is enabled by the following theorems. Table 1 depicts the notation required to analyze a sequence (tandem) of servers.

THEOREM 2. (Concatenation of Servers) *Consider a single flow f crossing a tandem of servers $s_1, \ldots, s_n$ where each s_i offers a service curve β_{s_i}. The overall service curve for f is their concatenation by convolution*

$$\beta_{s_1} \otimes \ldots \otimes \beta_{s_n} \quad = \quad \bigotimes_{i=1}^{n} \beta_{s_i}$$

THEOREM 3. (Left-Over Service Curve) *Consider a server s that offers a strict service curve β_s. Let s be crossed by two flow aggregates $\mathbb{F}_0$ and $\mathbb{F}_1$ with aggregate arrival curves $\alpha^{\mathbb{F}_0}$ and $\alpha^{\mathbb{F}_1}$, respectively. Then $\mathbb{F}_1$'s worst-case residual resource share under arbitrary multiplexing at s, i.e., its left-over service curve at s, is*

$$\beta_s^{\text{l.o.}\mathbb{F}_1} = \beta_s \ominus \alpha^{\mathbb{F}_0}$$

with $(\beta \ominus \alpha)(d) \coloneqq \sup\{0 \leq u \leq d \,|\, (\beta - \alpha)(u)\}$ denoting the non-decreasing upper closure of $(\beta - \alpha)(d)$.

Network Calculus Analysis

A network calculus analysis computes the delay bound for a specific flow (flow of interest, foi). From a conceptual point of view, two analysis steps can be distinguished [2, 3]. First, cross-traffic arrivals are bounded with Theorem 1, the output bound, such that the worst-case shape of cross-flows is known at the location of interference with the foi. This step reduces the network model required to derive the flow of interest's delay from the entire one to just a tandem of servers traversed by it. The foi's end-to-end delay bound in the feed-forward network can now be calculated with a less complex *tandem analysis*. This constitutes the second step of a DNC network analysis. It has seen much treatment in the literature in order to improve the ability to capture flow scheduling and cross-traffic multiplexing effects and thus provide more accurate delay bounds:

SFA (PBOO) [13]. Algebraic DNC tandem analyses result in a left-over service curve used for delay bound derivation with Theorem 1. Currently, there are two alternatives to compute the left-over service curve of a tandem: The Separate Flow Analysis (SFA) and the Pay Multiplexing Only Once (PMOO) analysis. The SFA is a straight-forward, hop-by-hop application of Theorems 3 and 2: First subtract cross-traffic arrivals and then concatenate the left-over service curves. Deriving the delay bound with a single, end-to-end left-over service curve will consider the flow of interest's burst term only once. This effect is therefore called Pay Bursts Only Once (PBOO). However, for cross-flows present at multiple consecutive hops, their bursts appear multiple times in the PBOO left-over service curve derivation.

PMOO [18]. The PMOO analysis provides an alternative containing each burst term only once. Its left-over service curve derivation reverses the operations, i.e., it convolves the tandem of servers before subtracting cross-traffic. Due to this end-to-end approach for all flows on the analyzed tandem, the PMOO analysis was considered superior to SFA. Yet, [17] shows that the SFA can arbitrarily outperform a PMOO tandem analysis. Both algebraic analyses thus complement each other.

[17] also provides a left-over service curve derivation that outperforms both algebraic ones. It transforms the network calculus system description to an optimization problem; departing from the algebraic methodology. Based on this insight, an optimization-based tight approach for delays in feed-forward was proposed in [5], however, it is NP hard and a computationally efficient solution for tight feed-forward analysis is currently not known. Therefore, we aim to improve the accuracy of algebraic network calculus.

3. ARRIVAL BOUNDING

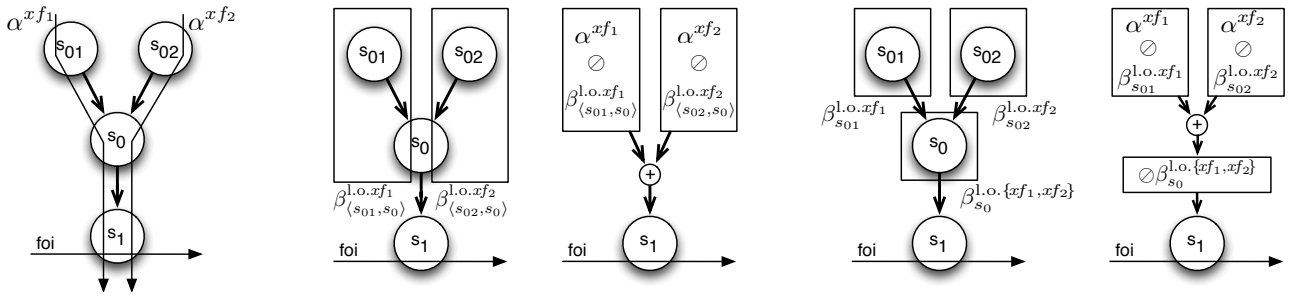
In this section, we focus on the first step of feed-forward network analysis with network calculus: Bounding cross-traffic arrivals. First, we provide the existing procedure. Then, we contribute our new method and prove its superiority when abstracting the feed-forward network to the analyzed flow's tandem of servers.

3.1 Segregated Cross-flow Bounding

Network calculus analysis previously focused on improved end-to-end semantics in the second step in order to tighten the delay bound. The state-of-the-art "generic way to compute performance bounds" in feed-forward networks is in line with this approach. It derives the cross-traffic arrival bound "for each flow at each system" [4], i.e., cross-flows are segregated from each other such that every cross-flow can be analyzed with a maximal end-to-end (i.e., source-to-interference location) semantic. The arrival bounding procedure of [4] is a straight-forward application of basic network calculus results that define the separate flow analysis (SFA): Flow segregation (Theorem 3), output bound derivation (Theorem 1), and output bound aggregation (Definition 5). Figures 2a and 2b provide a graphical illustration of these steps.

3.2 Aggregate Arrival Bounding

We propose to recursively apply Theorem 1, output bound, to (not segregated) cross-traffic aggregates – moving along the topology, starting from the location of interference with the flow of interest and stopping at the sources of cross-flows.



(a) Sample server graph. (b) Segregated flow arrival bounding. (c) Aggregate cross-traffic arrival bounding.

Figure 2: Decomposition of a server graph (a) for cross-traffic arrival bounding: (b) depicts the alternative to segregate the cross-flows [4] and (c) illustrates our new approach that maximizes cross-traffic aggregation during the arrival bounding.

This method calculates the arrival bound with the objective to maximize aggregation of flows, in fact deviating from the end-to-end analysis of cross-flows. Thus, the aggregate arrival bounding method defines a different decomposition of the server graph into left-over service curves than the state-of-the-art procedure from [4]. Figure 2c and the following algorithm depict our approach.

ALGORITHM 1. (Aggregate Arrival Bounding) *The foi's cross-traffic arrival bound at server s is derived as follows. Starting from s, paths of cross-flows are backtracked via links; each link l connects a source server l^{src} and a destination l^{dest}. The function $dest(s)$ returns the set of links whose destination is s, $x\left(foi,l\right)$ returns the cross-flows of foi on l and for a set of links $\mathbb{L}$ we define $x\left(foi,\mathbb{L}\right)=\bigcap_{l\in\mathbb{L}} x\left(foi,l\right)$. We get $x\left(x\left(\text{foi},l\right)\right)=F\left(l^{\text{src}}\right)\setminus x\left(\text{foi},l\right)$. Note, that flow sets might be restricted by location, e.g., $\alpha_{s_x}^{x(foi)}=\alpha_{s_x}^{x(foi)\cap F(s_x)}$.*

$$
\begin{aligned}
\alpha_s^{x(\text{foi})} &= \sum_{l_1\in dest(s)} \left(\alpha_{l_1^{\text{src}}}^{x(\text{foi},l_1)} \oslash \beta_{l_1^{\text{src}}}^{\text{l.o.}x(\text{foi},l_1)}\right) + \alpha^{F_{\text{src}}(s)\cap x(\text{foi})} \\
&= \sum_{l_1\in dest(s)} \left(\alpha_{l_1^{\text{src}}}^{x(\text{foi},l_1)} \oslash \left(\beta_{l_1^{\text{src}}} \ominus \alpha_{l_1^{\text{src}}}^{x(x(\text{foi},l_1))}\right)\right) \\
&\quad + \alpha^{F_{\text{src}}(s)\cap x(\text{foi})} \\
&= \sum_{l_1\in dest(s)} ((\sum_{l_2\in dest\left(l_1^{\text{src}}\right)} \left(\alpha_{l_2^{\text{src}}}^{x(\text{foi},\{l_2,l_1\})} \oslash \beta_{l_2^{\text{src}}}^{\text{l.o.}x(\text{foi},\{l_2,l_1\})}\right) \\
&\qquad\qquad + \alpha^{F_{\text{src}}(l_1^{\text{src}})\cap x(\text{foi},l_1)}) \\
&\quad \oslash \left(\beta_{l_1^{\text{src}}} \ominus \alpha_{l_1^{\text{src}}}^{x(x(\text{foi},l_1))}\right)) \\
&\quad + \alpha^{F_{\text{src}}(s)\cap x(\text{foi})} \\
&= \sum_{l_1\in dest(s)} ((\sum_{l_2\in dest\left(l_1^{\text{src}}\right)} \left(\alpha_{l_2^{\text{src}}}^{x(\text{foi},\{l_2,l_1\})} \oslash \beta_{l_2^{\text{src}}}^{\text{l.o.}x(\text{foi},\{l_2,l_1\})}\right) \\
&\qquad\qquad + \alpha^{F_{\text{src}}(l_1^{\text{src}})\cap x(\text{foi},,l_1)}) \\
&\quad \oslash (\beta_{l_1^{\text{src}}} \ominus \\
&\qquad \sum_{l_2\in dest\left(l_1^{\text{src}}\right)} \left(\alpha_{l_2^{\text{src}}}^{x(x(\text{foi},\{l_2,l_1\}))} \oslash \beta_{l_2^{\text{src}}}^{\text{l.o.}x(x(\text{foi},\{l_2,l_1\}))}\right))) \\
&\quad + \alpha^{F_{\text{src}}(s)\cap x(\text{foi})}
\end{aligned}
$$

The recursion of Algorithm 1 is exemplarily unfolded to illustrate the backtracking of interference in a feed-forward network. Initially, the foi's cross-traffic $\alpha_s^{x(\text{foi})}$ is split into the sum of flow arrivals from incoming links and those originating at s itself, $\alpha^{F_{\text{src}}(s)\cap x(\text{foi})}$. Next, the left-over service curves at the sources of incoming links are derived in order to separate the foi's cross-traffic to bound $x(\text{foi},l_1)$ from its cross-traffic $x(x(\text{foi},l_1))$, i.e., there are two directions to further backtrack flows by unfolding the term. Step 3 is towards the sources of $x(\text{foi},l_1)$ and step 4 towards $x(x(\text{foi},l_1))$ – the latter recursively starts an independent worst-case cross-traffic arrival bounding for $x(x(\text{foi},l_1))$. Both steps require to track the flow paths for proper left-over service computation in a feed-forward network. The backtracking will eventually terminate at the flow's sources.

Next, we prove that our aggregate arrival bounding method outperforms the segregated cross-flow procedure.

THEOREM 4. (Accuracy of Aggregate Arrival Bounding) *The aggregate arrival bounding method derives more accurate bounds than the segregated flow arrival bounding procedure for $\alpha\in\mathcal{F}_{\text{TB}}$ and $\beta\in\mathcal{F}_{\text{RL}}$.*

PROOF. Without loss of generality (detailed explanation follows after the proof), we prove this statement by showing that there are no beneficial effects of bounding long tandems (Figure 2b) that are more advantageous than bounding both flows aggregately on shorter tandems (Figure 2c).

First, we derive the segregated cross-flow arrival bounds at s_1, i.e., $\alpha_{s_1}^{xf_n}$, $n\in\{1,2\}$, and aggregate the results to $\alpha_{s_1}^{\{xf_1,xf_2\}}=\alpha_{s_1}^{xf_1}+\alpha_{s_1}^{xf_2}$.

$$
\begin{aligned}
\alpha_{s_1}^{xf_n} &= \alpha_{s_{0n}}^{xf_n} \oslash \beta_{\langle s_{0n},s_0\rangle}^{\text{l.o.}xf_n} \\
&= \alpha_{s_{0n}}^{xf_n} \oslash \left(\beta_{s_{0n}} \otimes \left(\beta_{s_0} \ominus \alpha_{s_0}^{xf_{\overline{n}}}\right)\right) \\
&= \alpha_{s_{0n}}^{xf_n} \oslash \left(\beta_{s_{0n}} \otimes \left(\beta_{s_0} \ominus \left(\alpha_{s_{0\overline{n}}}^{xf_{\overline{n}}} \oslash \beta_{s_{0\overline{n}}}^{\text{l.o.}xf_{\overline{n}}}\right)\right)\right) \\
&= \alpha_{s_{0n}}^{xf_n} \oslash \left(\beta_{s_{0n}} \otimes \left(\beta_{s_0} \ominus \left(\alpha_{s_{0\overline{n}}}^{xf_{\overline{n}}} \oslash \beta_{s_{0\overline{n}}}\right)\right)\right)
\end{aligned}
$$

where $\overline{n}$ denotes the opposite cross-flow's index, i.e., $\overline{1}=2$ and $\overline{2}=1$. Therefore,

$$
\begin{aligned}
\alpha_{s_0}^{\{xf_1,xf_2\}} &= \alpha_{s_{01}}^{xf_1} \oslash \left(\beta_{s_{01}} \otimes \left(\beta_{s_0} \ominus \left(\alpha_{s_{02}}^{xf_2} \oslash \beta_{s_{02}}\right)\right)\right) \\
&\quad + \alpha_{s_{02}}^{xf_2} \oslash \left(\beta_{s_{02}} \otimes \left(\beta_{s_0} \ominus \left(\alpha_{s_{01}}^{xf_1} \oslash \beta_{s_{01}}\right)\right)\right) \quad (1)
\end{aligned}
$$

Next, we derive the aggregate cross-traffic arrival bound $\alpha_{s_0}^{\{xf_1,xf_2\}}$ according to Algorithm 1.

$$
\begin{aligned}
\alpha_{s_0}^{\{xf_1,xf_2\}} &= \left(\alpha_{s_0}^{xf_1} + \alpha_{s_0}^{xf_2}\right) \oslash \beta_{s_0}^{\text{l.o.}\{xf_1,xf_2\}} \\
&= \left(\left(\alpha_{s_{01}}^{xf_1} \oslash \beta_{s_{01}}\right) + \left(\alpha_{s_{02}}^{xf_2} \oslash \beta_{s_{02}}\right)\right) \oslash \beta_{s_0} \quad (2)
\end{aligned}
$$

Last, we show that the former arrival bound cannot be smaller than the latter one, i.e., (2) $\leq$ (1) always holds. According to [3], Lemma 12, we can distribute the deconvolution of token-bucket arrivals with a rate-latency service curve over the aggregation. For (2), this means:

$$\left(\left(\alpha_{s01}^{xf_1} \oslash \beta_{s_{01}}\right) + \left(\alpha_{s02}^{xf_2} \oslash \beta_{s_{02}}\right)\right) \oslash \beta_{s_0} = \left(\alpha_{s01}^{xf_1} \oslash \beta_{s_{01}}\right) \oslash \beta_{s_0} + \left(\alpha_{s02}^{xf_2} \oslash \beta_{s_{02}}\right) \oslash \beta_{s_0}$$

and (2) $\leq$ (1) translates to

$$\begin{aligned}&\left(\alpha_{s01}^{xf_1} \oslash \beta_{s_{01}}\right) \oslash \beta_{s_0} + \left(\alpha_{s02}^{xf_2} \oslash \beta_{s_{02}}\right) \oslash \beta_{s_0} \\ &\leq \alpha_{s01}^{xf_1} \oslash \left(\beta_{s_{01}} \otimes \left(\beta_{s_0} \ominus \left(\alpha_{s02}^{xf_2} \oslash \beta_{s_{02}}\right)\right)\right) \\ &\quad + \alpha_{s02}^{xf_2} \oslash \left(\beta_{s_{02}} \otimes \left(\beta_{s_0} \ominus \left(\alpha_{s01}^{xf_1} \oslash \beta_{s_{01}}\right)\right)\right).\end{aligned}$$

We now compare each segregated cross-flow's impact on the final arrival bound. This gives us two sub-terms:

$$\begin{aligned}&\left(\alpha_{s0n}^{xf_n} \oslash \beta_{s_{0n}}\right) \oslash \beta_{s_0} \\ &\leq \alpha_{s0n}^{xf_n} \oslash \left(\beta_{s_{0n}} \otimes \left(\beta_{s_0} \ominus \left(\alpha_{s0\overline{n}}^{xf_{\overline{n}}} \oslash \beta_{s_{0\overline{n}}}\right)\right)\right)\end{aligned}$$

Next, we can apply the composition rule for $\oslash$ ([13], Theorem 3.1.12) and use $\otimes$'s commutativity to reformulate the terms to

$$\begin{aligned}&\alpha_{s0n}^{xf_n} \oslash \beta_{s_{0n}} \oslash \beta_{s_0} \\ &\leq \alpha_{s0n}^{xf_n} \oslash \beta_{s_{0n}} \oslash \left(\beta_{s_0} \ominus \left(\alpha_{s0\overline{n}}^{xf_{\overline{n}}} \oslash \beta_{s_{0\overline{n}}}\right)\right).\end{aligned}$$

These equations reveal that the crucial difference between both arrival bounding alternatives is the respective (left-over) service curve at server s_0. Only this single system service curve defines the difference between both arrival bounds; end-to-end effects cancel out entirely.

Finally, due to all curves being from $\mathcal{F}_0$, we know that

$$\beta_{s_0} \geq \beta_{s_0} \ominus \left(\alpha_{s0n}^{xf_n} \oslash \beta_{s_{0n}}\right),$$

i.e., the aggregate arrival boundings service curve is larger, and for $\forall \beta_{s_x}, \beta_{s_y} \in \mathcal{F}_{\text{RL}}$ and $\forall \alpha \in \mathcal{F}_{\text{TB}}$ in particular, we know from [3] that

$$\beta_{s_x} \geq \beta_{s_y} \quad \Rightarrow \quad (\alpha \oslash \beta_{s_x}) \leq \left(\alpha \oslash \beta_{s_y}\right),$$

i.e., a larger service curve leads to a smaller output bound.

Therefore, aggregate cross-traffic arrival bounding outperforms segregated cross-flow arrival bounding. □

Note, that equality between both arrival bounding alternatives only holds for the trivial cases where the service β_{s_0} is infinitely large, i.e., $\beta_{+\infty,0}$, or $\alpha_{s0n}^{xf_n}$ (α^{xf_n}) is zero.

Last, let us clarify why the seemingly simple scenario of Figure 2a allows for our contribution's generality. Any more involved server graph can be decomposed into a combination of variants of the graph shown in Figure 2a while retaining Theorem 4's validity:

Intermediate tandems instead of single servers

s_0, s_{01}, or s_{02} can be assumed to consist of multiple servers in tandem that were convolved into single servers for analysis (see Theorem 2). Then, the above proof as well as Algorithm 1 virtually move across the server graph tandem-by-tandem instead of hop-by-hop.

Cross-traffic of cross-traffic

Aggregate arrival bounding compromises on the source-to-interference view of segregated cross-flow arrival bounding. Tandems are restricted to sequences of servers shared by all flows in the respective cross-traffic aggregate. If an aggregate had its own cross-traffic (see $x(x(\text{foi}, l_1))$ above), another left-over service curve derivation and thus an arrival bounding (of $x(x(\text{foi}, l_1))$) would be needed. This derivation operates with its own worst-case assumptions for $x(x(\text{foi}, l_1))$. In the best case for segregated arrival bounding, when it is able to derive the same $\alpha^{x(x(\text{foi},l_1))}$ as aggregate arrival bounding, Theorem 4's proof remains unchanged. In any other case, it even operates on worse left-over service curves.

Further cross-traffic with the same interference pattern

Assume another cross-flow (aggregate) xf_3 merges with the existing cross-traffic $\{xf_1, xf_2\}$ at server s_0, entering from a different server (or convolved tandem of left-over service curves) s_{03}. Then, equations containing the segregated flows' impact on arrival bounds are expanded with xf_3's influence

$$\begin{aligned}&\alpha_{s01}^{xf_1} \oslash \beta_{s_{01}} \oslash \beta_{s_0} \\ &\geq \alpha_{s01}^{xf_1} \oslash \beta_{s_{01}} \oslash \left(\beta_{s_0} \ominus \left(\alpha_{s02}^{xf_2} \oslash \beta_{s_{02}} + \alpha_{s03}^{xf_3} \oslash \beta_{s_{03}}\right)\right), \\ &\alpha_{s02}^{xf_2} \oslash \beta_{s_{02}} \oslash \beta_{s_0} \\ &\geq \alpha_{s02}^{xf_2} \oslash \beta_{s_{02}} \oslash \left(\beta_{s_0} \ominus \left(\alpha_{s01}^{xf_1} \oslash \beta_{s_{01}} + \alpha_{s03}^{xf_3} \oslash \beta_{s_{03}}\right)\right)\end{aligned}$$

and a similar equation for xf_3's impact on the aggregate arrival bound is added

$$\begin{aligned}&\alpha_{s03}^{xf_3} \oslash \beta_{s_{03}} \oslash \beta_{s_0} \\ &\geq \alpha_{s03}^{xf_3} \oslash \beta_{s_{03}} \oslash \left(\beta_{s_0} \ominus \left(\alpha_{s01}^{xf_1} \oslash \beta_{s_{01}} + \alpha_{s02}^{xf_2} \oslash \beta_{s_{02}}\right)\right).\end{aligned}$$

Neither adaptation impacts the proof's core statement; s_0 still constitutes the crucial bottleneck server where aggregation outperforms segregation.

Further cross-flows with different interference patterns

Assume s_0, s_{01}, and s_{02} actually consisted of tandems of servers and there were further cross-flows of the foi, i.e., in $x(\text{foi})$, merging somewhere on these tandems. That is, the server graph consists of a bigger, more complex feed-forward network than depicted in Figure 2a. In this case, the above reasoning would have to be repeated recursively for every backtracking required – similar to *cross-traffic of cross-traffic*. Therefore, each recursion level has its own bottleneck server where aggregation outperforms segregation and the effect causing aggregation's superiority is amplified.

3.3 Exhaustive Aggregate Arrival Bounding

Algorithm 1 defines a hop-by-hop method where left-over service curves of individual servers are derived, similar to the segregated arrival bounding procedure, yet, for cross-traffic aggregates. While this exploits the PBOO-effect of SFA, additional gains from the PMOO-effect can only be achieved if the left-over service curve is derived for tandems of multiple consecutive hops, i.e., in a tandem-by-tandem procedure. In this section, we extend our aggregate arrival bounding to gain from both, the PBOO and the PMOO effect, in a single, exhaustive method.

The prerequisite for tandem analysis during aggregate arrival bounding is that aggregated flows cross multiple consecutive hops. For instance, let s_0 in Figure 2a be a system

of two servers s_x and s_y, both crossed by xf_1 and xf_2 in this order (cf. *Intermediate tandems instead of single servers*). Then, we can transform Algorithm 1's proceeding to

$$\begin{aligned}\alpha_{s_1}^{\{xf_1,xf_2\}} &= \alpha_{s_x}^{\{xf_1,xf_2\}} \oslash \beta_{s_x}^{\text{l.o.}\{xf_1,xf_2\}} \oslash \beta_{s_y}^{\text{l.o.}\{xf_1,xf_2\}} \\ &= \alpha \oslash \left(\beta_{s_x}^{\text{l.o.}\{xf_1,xf_2\}} \otimes \beta_{s_y}^{\text{l.o.}\{xf_1,xf_2\}}\right) \\ &= \alpha \oslash \beta_{\langle s_x,s_y\rangle}^{\text{PBOO l.o.}\{xf_1,xf_2\}},\end{aligned}$$

where $\beta^{\text{PBOO l.o.}}$ denotes the hop-by-hop left-over service curve on a tandem of servers. In order to exploit the PMOO-effect, we replace $\beta^{\text{PBOO l.o.}}$ with its PMOO analysis counterpart $\beta^{\text{PMOO l.o.}}$.

From [17] we know that neither of the two algebraic alternatives to derive the left-over service curve strictly outperforms the other one in any server graph and flow entanglement. Moreover, their results depend on actual parameters of service curves and cross-traffic arrival bounds. Whereas the service curves are known, the arrival bounds need to be computed first. When decomposing the server graph into left-over service derivations, the superior $\beta^{\text{l.o.}}$ is thus still unknown. For this reason, we propose to apply both alternative derivations in parallel on each level of the recursion and finally check every combination for its result in order to find the best delay bound. We call this method the exhaustive aggregate arrival bounding. It is illustrated in Algorithm 2.

The exhaustive aggregate arrival bounding algorithm depicts the DiscoDNC v2 implementation [2]. Lowercase letters denote variables, blackboard bold letters denote sets of variables, and the algorithm proceeds as follows:

Algorithm 2: Exhaustive arrival bound computation.

Input : Set of flows $\mathbb{F}$ to bound at server s
Output: Set of arrival bounds $\mathbb{A}_s^{\mathbb{F}}$

```
1  ArrivalBoundsAtServer(Server s, Flow set F)
2      foreach Link l ∈ s.getInLinks() do
3          F_l = F ∩ F(l);
4          A_l^F = ArrivalBoundsOnLink(l, F_l);
5          A_inlinks^F.put(l, A_l^F);
6      end
7      A_s^F = getABCombinations(A_inlinks^F);
8      foreach α ∈ A_s^F do
9          α += F_src(s) ∩ F;
10     end
11 return A_s^F;
12 ArrivalBoundsOnLink(Link l, Flow set F)
13     T_shared = getSharedPathTo(F, l.getSource());
14     foreach Server s ∈ T_shared do
15         F_s^{x(F)} = F(s) \ F;
16         A_{T_shared}^{x(F)}.put(s, ArrivalBoundsAtServer(F_s^{x(F)}));
17     end
18     B_{T_shared}^{l.o.F} = LeftOverBetas(T_shared, A_{T_shared}^{x(F)});
19     source = T_shared.getSource();
20     A_source^F = ArrivalBoundsAtServer(source, F);
21     foreach β ∈ B_{T_shared}^{l.o.F} do
22         A_l^F.addAll(A_source^F ⊘ β);
23     end
24 return A_l^F;
```

The objective is similar to Algorithm 1 and so is the entry point – a set of flows $\mathbb{F}$ to bound and a server s to bound their arrivals at. Like in Algorithm 1, we first iterate over the inlinks of s (lines 2 to 6) and then add the arrivals of flows originating at s (lines 8 to 10). A difference to Algorithm 1 can be found in line 7: The exhaustive aggregate arrival bounding returns at least two bounds per inlink, a PBOO and a PMOO one. The exact amount depends on the tandems in the recursion. Therefore, all combinations of different arrival bounds for all inlinks are valid bounds.

Inlinks define the locations where recursive cross-traffic bounding may be required or where an analyzed cross-flow aggregate may be forced to be split into sub-aggregates (e.g., inlinks of s_0 in Figure 2 or the summation over inlinks when unfolding the term in Algorithm 1). In both cases PBOO and PMOO arrival bounds need to be derived next. The effort involved thus grows with the number of links this happens, yet, an efficient implementation that convolves alternative arrival bounds into a single one that is also cached for reuse keeps the overhead marginal and allows for evaluations of the size shown in Section 4. For every inlink, the arrival bound is computed by first searching for the longest tandem flows traverse aggregately ($\mathbb{T}_{\text{shared}}$, line 13), then recursively deriving the cross-traffic arrivals for this tandem (lines 14 to 17), as well as applying the PBOO and PMOO left-over service curve derivation (line 18) and computing the flow aggregate's output from $\mathbb{T}_{\text{shared}}$ (lines 19 to 24).

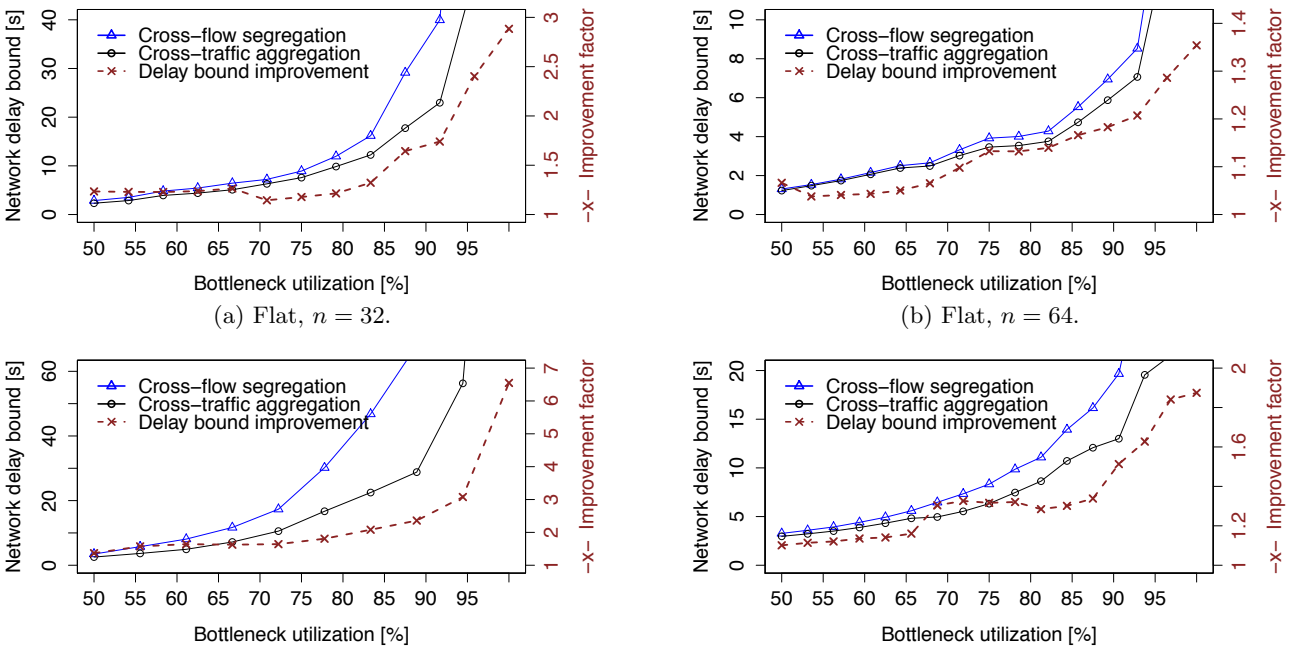
4. EVALUATION

In this section, we evaluate the impact of our exhaustive aggregate cross-traffic arrival bounding method on the worst-case end-to-end delay bound occurring in networks with larger server graphs than the one depicted in Figure 2a. We call this worst case the *network delay bound* as it is globally valid for all flows in the analyzed network. Our evaluation extensively quantifies Section 3.2's last remark: Amplification of aggregate bounding's superiority when the number of bottlenecks as well as their utilization increases.

Network Calculus Evaluation

First, let us explain the best practice to analyze arbitrary networks with network calculus: As shown in Section 2.1, we start from a network graph representing the topology, e.g., generated by an existing topology generator. We then derive its according server graph and assign service curves to servers. In this paper, we set all service curves β to the rate function $\beta_{100\text{Mbps},0}$ to resemble 100Mbps Ethernet links. A network calculus analysis as depicted in Section 2.2 requires flows to not have cyclic dependencies. By applying turn prohibition [19], the server graph becomes feed-forward and thus free of cyclic dependencies between flows[1]. Flows naturally connect a pair of network devices, yet, for analysis they need to be routed through the turn-prohibited server graph. According to our objective to derive the network delay bounds with increasing bottleneck utilization, we randomly choose each flow's source and sink device, route on the shortest path in the server graph and assign unit sized arrival curves $\alpha = \gamma_{1\text{Mbps},1\text{Mb}}$. The amount of flows is continuously increased until reaching the network's capacity limit. Each

[1] Alternatively, a fixed-point analysis can be carried out [12].



(a) Flat, $n = 32$.

(b) Flat, $n = 64$.

(c) Hierarchical, $n = 32$.

(d) Hierarchical, $n = 64$.

Figure 3: Network delay bounds for flat and hierarchical Erdős-Rényi networks with $n \in \{32, 64\}$ devices.

flow configuration is analyzed with both algebraic network calculus analyses, SFA and PMOO, to obtain every flow's best delay bound – with either arrival bound alternative of Section 3. Finally, result plots depict the derived network delay bounds for bottleneck utilizations between 50% and 100%, i.e., between half the maximum amount of flows the network can route with delay guarantees and the maximum itself. Our result plots additionally contain the achieved improvement factor $\frac{D(\text{segregated PBOO})}{D(\text{exhaust. aggregate})}$ suggested by [5].

Numerical Experiments

In our numerical experiments, we use the aSHIIP topology generator [21] to randomly create Erdős-Rényi network graphs following the $G(n, p)$-model with $p = 0.1$. We evaluate the impact of improved arrival bounding in flat and hierarchical network topologies with 32 and 64 devices resulting in 114 and 414 servers (flat), respectively, and 73 as well as 394 servers (hierarchical).

In flat networks, our exhaustive aggregate arrival bounding achieves a considerable delay bound improvement in the small example (see Figure 3a, dashed brown line with crosses, right y-axis). Even when the network sees a small bottleneck utilization of 50%, we achieve improvements of factor 1.23, which already reduces the delay bound by about 18%. With growing utilization and fast growing delay bounds, the factor increases to 2.88. Increasing the network size, flow density decreases such that unfortunate entanglements creating severe bottlenecks become less likely. Therefore, the amount of bottlenecks as well as the impact of the exhaustive aggregate arrival bounding decreases. Nonetheless, the improvement remains considerable, especially when the bottleneck utilization limit is approached (factor 1.354, delay bound reduction of 26%, Figure 3b).

Hierarchical networks possess a set of predefined bottleneck links – those links connecting the levels of the hierarchy. The aSHIIP generator generates a relatively static amount of levels, 5 to 6 in our experiments. Randomly routed flows are thus more likely to cross levels in the small network where we can observe considerable gains at low utilizations as well as a higher maximum (factor 6.55, −84%, Figure 3c) than in the large network (factor 1.875, −46%, Figure 3d). In both cases, the predefined bottlenecks of the hierarchical topology lead to a higher impact of exhaustive aggregate arrival bounding than in the flat network topologies.

Our experiments reveal the following trends:

- The improved arrival bounding method presented in this paper allows to compute more accurate network delay bounds, independent of the bottleneck utilization. The highest gains are achieved when the network reaches its capacity limit.
- The cross-traffic arrival bounding method itself does not affect the network delay bound's asymptotic growth pattern when utilization is increased. However, the growth rate is slowed down considerably and in cases of a large bottleneck utilization, our aggregate arrival bounding can achieve network delay bounds that are multiple times more accurate than the existing ones.

AFDX Case Study

Last, we evaluate the industrial AFDX (Avionics Full-Duplex Switched Ethernet) network topology. We create the network graph according to the scheme provided in [6]: It consists of 16 switches at the core and 125 end-systems connected with 100Mbps Ethernet links in order to simulate the network deployed in the Airbus A380.

According to the current AFDX specification, flows are routed within so-called virtual links (VLs). Each VL connects a single source end-system to multiple sink end-systems with fixed resource reservation on the path between these systems. In the view of the network calculus analysis, VLs correspond to multicast flows that reserve large resource

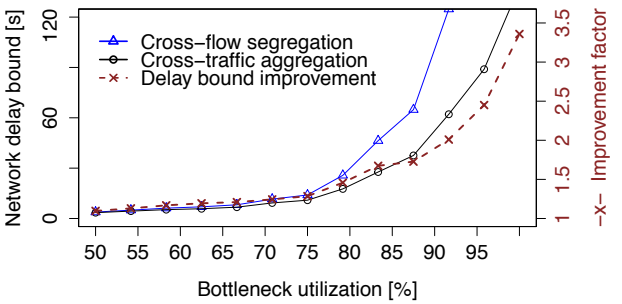


Figure 4: Network delay bounds in the AFDX topology.

shares. An examination of the problems due to VLs' coarse granularity can be found in [15]. Moreover, network calculus does not provide a specialized analysis for multicast communication. The analysis treats each multicast flow as a set of independent unicast flows; one for every source-sink pair of connected network devices. For these reasons, we retain our evaluation approach and restrict to the immutable part of AFDX: The deployed networks' common topology design.

The AFDX topology can benefit much from our method – Figure 4 illustrates the gap between old and new network delay bound. Starting at an improvement factor of 1.1, it grows fast until reaching the bottleneck capacity limit and factor 3.36. Each of the 250 links to and from an end-system gets congested easily, similar to the few bottlenecks connecting different levels in the hierarchical Erdős-Rényi networks. Moreover, AFDX's flat core is very small and thus prone to the dynamic bottleneck emergence already observed in the flat Erdős-Rényi networks. Therefore, the results are particularly pronounced in the AFDX topology: Although the analyzed network is larger than the 64-devices networks above, analysis results are most similar to the flat Erdős-Rényi network with 32 network devices. The network delay bound as well as the improvement both grow exponentially with the bottleneck utilization; aggregate cross-traffic arrival bounding reduces the network delay bound computed by the cross-flow the segregation procedure by 70%.

5. CONCLUSION

In this paper, we present a new method to bound cross-traffic arrivals. It maximizes flow aggregation, yet, compromises on the length of tandems to analyze – the previous key objective of DNC analysis. Nonetheless, we prove that, with token-bucket arrivals and rate-latency service, our method outperforms the previous procedure – even by considerable margins as our evaluation results show.

While we focus on the analysis of general feed-forward networks, we also restricted our presentation by only covering arbitrary multiplexing servers. Nonetheless, the aggregate arrival bounding can also be applied to FIFO multiplexing servers. To be precise, the PBOO and PMOO left-over service curve derivations can be replaced with, e.g., the least upper delay bound method that derives left-over service curves for tandems of FIFO multiplexing servers [14].

6. REFERENCES

[1] S. Bondorf and J. B. Schmitt. Statistical Response Time Bounds in Randomly Deployed Wireless Sensor Networks. In *Proc. IEEE LCN*, 2010.

[2] S. Bondorf and J. B. Schmitt. The DiscoDNC v2 – A Comprehensive Tool for Deterministic Network Calculus. In *Proc. ValueTools*, 2014.

[3] S. Bondorf and J. B. Schmitt. Boosting Sensor Network Calculus by Thoroughly Bounding Cross-Traffic. In *Proc. IEEE INFOCOM*, 2015.

[4] A. Bouillard. *Algorithms and Efficiency of Network Calculus*. Habilitation thesis, ENS, 2014.

[5] A. Bouillard, L. Jouhet, and E. Thierry. Tight Performance Bounds in the Worst-Case Analysis of Feed-Forward Networks. In *Proc. INFOCOM*, 2010.

[6] M. Boyer, N. Navet, and M. Fumey. Experimental Assessment of Timing Verification Techniques for AFDX. In *Proc. ERTS*, 2012.

[7] C.-S. Chang. *Performance Guarantees in Communication Networks*. Springer, 2000.

[8] R. F. Coelho, G. Fohler, and J.-L. Scharbarg. Worst-Case Backlog for AFDX Network with n-Priorities. In *Proc. RTN Workshop*, 2014.

[9] R. L. Cruz. A Calculus for Network Delay, Part I: Network Elements in Isolation. *IEEE Transactions on Information Theory*, 1991.

[10] F. Frances, C. Fraboul, and J. Grieu. Using Network Calculus to Optimize AFDX Network. In *ERTS*, 2006.

[11] T. Hamza, J.-L. Scharbarg, and C. Fraboul. Priority Assignment on an Avionics Switched Ethernet Network (QoS AFDX). In *Proc. IEEE WFCS*, 2014.

[12] B. Jonsson, S. Perathoner, L. Thiele, and W. Yi. Cyclic Dependencies in Modular Performance Analysis. In *Proc. ACM EMSOFT*, 2008.

[13] J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001.

[14] L. Lenzini, E. Mingozzi, and G. Stea. End-to-end Delay Bounds in FIFO-multiplexing Tandems. In *Proc. ValueTools*, 2007.

[15] R. Mancuso, A. V. Louis, and M. Caccamo. Using Traffic Phase Shifting to Improve AFDX Link Utilization. In *Proc. ACM EMSOFT*, 2015.

[16] J. B. Schmitt, N. Gollan, S. Bondorf, and I. Martinovic. Pay Bursts Only Once Holds for (Some) non-FIFO Systems. In *Proc. IEEE INFOCOM*, 2011.

[17] J. B. Schmitt, F. A. Zdarsky, and M. Fidler. Delay Bounds under Arbitrary Multiplexing: When Network Calculus Leaves You in the Lurch ... In *Proc. IEEE INFOCOM*, 2008.

[18] J. B. Schmitt, F. A. Zdarsky, and I. Martinovic. Improving Performance Bounds in Feed-Forward Networks by Paying Multiplexing Only Once. In *Proc. GI/ITG MMB*, 2008.

[19] D. Starobinski, M. Karpovsky, and L. A. Zakrevski. Application of Network Calculus to General Topologies Using Turn-Prohibition. *IEEE/ACM Transactions on Networking*, 2003.

[20] T. Steinbach, H.-T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz. Beware of the Hidden! How Cross-traffic Affects Quality Assurances of Competing Real-time Ethernet Standards for In-Car Communication. In *Proc. IEEE LCN*, 2015.

[21] J. Tomasik and M.-A. Weisser. Internet Topology on AS-level: Model, Generation Methods and Tool. In *Proc. IEEE IPCCC*, 2010.