Network Synthesis under Delay Constraints: The Power of Network Calculus Differentiability

Fabien Geyer*†
*Airbus Central R&T, Munich, Germany
†Technical University of Munich, Munich, Germany

Steffen Bondorf‡
‡Faculty of Computer Science, Ruhr University Bochum, Germany

***Abstract*—With the advent of standards for deterministic network behavior, synthesizing network designs under delay constraints becomes the natural next task to tackle. Network Calculus (NC) has become a key method for validating industrial networks, as it computes formally verified end-to-end delay bounds. However, analyses from the NC framework were thus far designed to bound one flow's delay at a time. Attempts to use classical analyses for derivation of a network configuration revealed this approach to be poorly fitted for practical use cases. Take finding a delay-optimal routing configuration: One model for each routing alternative had to be created, then each flow delay had to be bounded, then the bounds were compared to the given constraints. To overcome this three-step procedure, we introduce Differential Network Calculus. We extend NC to allow for differentiation of delay bounds w.r.t. to a wide range of network parameters – such as flow routes. This opens up NC to a class of efficient nonlinear optimization techniques taking advantage of the delay bound computation's gradient. Our numerical evaluation on the routing problem shows that our novel method can synthesize flow path in a matter of seconds, outperforming existing methods by multiple orders of magnitude.**

I. Introduction

With the current developments of networking solutions for strict reliability and safety requirements (such as IEEE Time Sensitive Networking (TSN)), formal verification and optimization of safety-critical networks has become an important step of the design process in various industries [1]. While using mathematical models and formalization of end-to-end delay bounds has now become common practice, optimizing and fine-tuning networks under such formulations remains a difficult task. This main difficulty arises from the inherent combinatorial property and nonlinearity of the formal models, making them hard problems to solve in polynomial time. Previous attempts were often limited to small networks.

In this paper, we propose a novel approach for modeling, optimizing and synthesizing networks under hard end-to-end delay constraints able to scale to networks of realistic sizes. We introduce an approach able to efficiently synthesize flows' paths, flows' priorities, and schedulers' parameters. We bound end-to-end delays using Network Calculus (NC) based on the (min,plus) algebra [2]. While this method is commonly used in some industries for formally validating delay requirements, it is rarely used for synthesis or as a design tool. Existing NC analyses were created to analyze already complete network designs, making them only suitable for a design space exploration that enumerates and ranks different designs.

We present an extension of NC called Differential Network Calculus (DiffNC). We formally show that under the assumptions traditionally used for validating industrial networks (i.e. token-bucket and rate-latency curves), a flow's delay bound computed using the (min,plus) algebra is differentiable according to the different curves' parameters in the network. This enables a wide range of applications, among which is gradient-based nonlinear programming (NLP). Via variable relaxation, we demonstrate that traditional NLP methods based on Newton's method can efficiently solve the aforementioned network optimization problems – and synthesize configurations. We show that these optimization methods are highly efficient, scale well and provide the best solutions, making them applicable to networks of sizes found in the industry.

In the realm of NC, previous works already formalized NC as an optimization problem, by proposing a formulation of the end-to-end delay bounds as a linear program (LP) [3]. This approach is able to achieve tight delay bounds. We illustrate that this LP formulation can be extended to multiple flows in its objective function, too. It can be used to optimize paths of flows, yet, the objective function suffers from poor expressivity for some important types of constraints. Additionally, we show that it suffers from poor scalability, taking more than one hour of computation even on relatively small networks. Its limitations render the approach unsuitable for realistic problems.

Our proposed approach has the following benefits. First, we use an existing NC analysis to derive an (min,plus)-algebraic term bounding the delay, yet with integer variables encoding alternatives like potential flow paths. Then, for finding the best alternative with NLP, we may include nonlinear constraints and nonlinear objective functions, enabling for concepts like utility functions [4] on the delay bounds, or even reducing the tail of the delay bound distribution. Finally, we illustrate that our approach scales to networks with up to 1000 flows, a size similar to industrial use-cases [5, 6, 7]. Our implementation is based on efficient computer algebra system (CAS) and automatic differentiation (AD), enabling us to efficiently compute the end-to-end delay bounds and their gradient without paying dearly in terms of computation times. As an application of our approach, we illustrate how to use DiffNC for finding the best priorities and certain scheduler parameters in TSN networks.

This paper is organized as follows: Section II presents the related work, followed by NC in Section III. Section IV presents the mathematical foundations of DiffNC and suitable

network optimization problems. Section V extends existing LP-based NC to compete with DiffNC. We numerically evaluate and compare DiffNC against other optimization methods in Section VI. Finally, we discuss applications of DiffNC to, e.g., TSN, in Section VII and Section VIII concludes the paper.

II. Related work

Optimization with delay bounds: Various works already investigated delay bound minimization. [8] proposed one of the early works on route optimization based on NC by using shortest path on graphs with weights set according to delay bounds. While this approach was shown to be efficient, it is limited to the optimization of a single flow's route, restricting its use when multiple routes need to be optimized.

In [9], various iterative algorithms for rerouting flows to minimize tail delays were detailed. These NC-based algorithms appeared to scale to realistic network sizes. Various works modeled delay bounds as an integer linear programming (ILP) for optimizing routes. [10] used a linear formulation of the NC end-to-end delay bound for optimizing the network-on-chip of a many core processor. Their approach showed promising results compared to a nonlinear formulation on a small network. Similarly, [11] recently proposed another ILP formulation tailored to TSN and multicast flows, optimizing paths and schedules. For both [10] and [11], the scalability of both approaches to larger networks remains unclear.

In the scope of TSN, [12] applied worst-case delay calculations in combination with a greedy optimization approach. While the results show improvements over a shortest path approach, the formulation is tailored to the TSN schedulers and the optimality of the solution is difficult to assess.

Derivation of service requirements: An NC-based approach that can derive a lower bound on the system service was proposed in [13, 14]. It extends the (min,plus)-algebraic NC with novel theory to take as input a function upper bounding the delay to be guaranteed under a certain load level. However, the approach is currently restricted to FIFO systems. DiffNC can perform the same task, yet without any restricting assumptions. It can fully use existing algebraic NC theory and analyses.

NC combined with other methods: (min,plus) algebra can be replaced with (max,plus) to better fit discrete event systems [15]. NC was paired with event stream theory [16] and with timed automata [17] for state-based system modeling. NC has been applied to the component-based models of real-time systems [18], giving rise to the so-called real-time calculus.

Various formulations of the (min,plus) algebra as LP were proposed, either addressing networks without assumptions on the multiplexing of flows [3], or with FIFO scheduling [19]. These formulations provide tight delay bounds but scale poorly, as shown by [20] and later also in Section V. These concerns were partially addressed recently in [21]. Additionally, [22] also proposed an ILP for optimizing time-division multiple access (TDMA) schedules in combination with NC.

Finally, machine learning (ML) was recently brought to NC to speed-up costly network analyses originally requiring a search mimicking optimization in the algebraic approach. DeepTMA was proposed in [23, 24] as a framework for predicting the best contention model. Similarly, DeepFP [25] targeted the prediction of best flow prolongation. [26] applied similar deep learning techniques for checking feasibility of network configurations, yet not for their synthesis.

To the best of our knowledge, this is the first work investigating the differentiability of an end-to-end NC delay bound.

III. Deterministic Network Calculus

A. Network Calculus System Model [2]

NC models a network as a directed graph of connected queueing locations, the so-called server graph $\mathcal{G} = (\mathcal{S}, \mathcal{E})$. A server $s \in \mathcal{S}$ offers forwarding of queued data. Flows cross the graph $\mathcal{G}$ from a source server to a destination server, both in $\mathcal{S}$. Data put into the network by flows is only known at their resp. source, and is characterized by a cumulative function in

$$\mathcal{F}_0^+ = \left\{ f : \mathbb{R}^+ \to \mathbb{R}^+ \mid f(0) = 0,\ \forall s \le t\ :\ f(t) \ge f(s) \right\}. \quad (1)$$

When flows multiplex in a server's queue, we assume no knowledge about the resulting order (arbitrary multiplexing).

NC uses univariate functions that give deterministic bounds on either data arrivals or forwarding in Δ-time, called curves. Curves are in $\mathcal{F}_0$, an extension of $\mathcal{F}_0^+$ by $\forall t < 0 : f(t) = 0$.

Definition 1 (Arrival Curve): Given a flow f described by $A \in \mathcal{F}_0^+$, a function $\alpha \in \mathcal{F}_0$ is an arrival curve for f iff

$$\forall\, 0 \le d \le t\ :\ A(t) - A(t-d) \le \alpha(d). \quad (2)$$

The forwarding service offered by servers in $\mathcal{S}$ needs to be a lower bounding curve relating data output to data input.

Definition 2 (Service Curve): If a server receives a data input $A \in \mathcal{F}_0^+$ and produces an output $A' \in \mathcal{F}_0^+$, then it is said to offer service curve $\beta \in \mathcal{F}_0$ iff

$$\forall t : A'(t) \ge \inf_{0 \le d \le t} \{ A(t-d) + \beta(d) \}. \quad (3)$$

Definition 3 (Strict Service Curve): A server offers a strict service curve β if it produces an output of at least $\beta(d) \in \mathcal{F}_0$ during periods of queued data of length d.

B. Algebraic Network Calculus Analysis

An NC analysis computes a bound on the end-to-end delay of a specific flow of interest (foi) in a complete server graph model. We are concerned with the NC branch that derives a (min,plus)-algebraic term describing all flow interactions that impact the end-to-end delay under arbitrary multiplexing.

Definition 4 (Operations): Let functions $f, g \in \mathcal{F}_0$. Then

$$\text{aggregation: } (f+g)(d) = f(d) + g(d), \quad (4)$$

$$\text{convolution: } (f \otimes g)(d) = \inf_{0 \le u \le d} \{ f(d-u) + g(u) \}, \quad (5)$$

$$\text{deconvolution: } (f \oslash g)(d) = \sup_{u \ge 0} \{ f(d+u) - g(u) \}, \quad (6)$$

$$\text{left-over: } (f \ominus g)(d) = \sup_{0 \le u \le d} \{ f(u) - g(u) \}. \quad (7)$$

Algebraic NC analyses such as Separate Flow Analysis (SFA) and Pay Multiplexing Only Once (PMOO) require service curves to be strict and then give rules to arrange the

operations to form a delay bounding term. In order to do so, each of these analyses has a different procedure to backtrack crossflows to their sources [27], e.g. SFA proceeds server-by-server and backtracks one crossflow aggregate per server's incoming edge in $\mathcal{E}$. In the end, the (min,plus)-algebraic term describes a single end-to-end left-over service curve for the foi that is used to compute its delay bound:

Theorem 1 (Delay Bound): A flow with arrival curve α that crosses a server (or sequence of servers) offering service curve β to the flow experiences a delay bounded by

$$h(\alpha, \beta) := \inf \{d \geq 0 \mid (\alpha \oslash \beta)(-d) \leq 0\}. \tag{8}$$

IV. Differential Network Calculus

We describe here our extension of NC, called Differential Network Calculus (DiffNC). After a formalization of our mathematical framework, we describe one exemplary application: optimization of flow paths using NLP. Further applications of DiffNC are detailed later in Section VII, covering TSN-related topics (flow priorities and scheduler parameters).

A. From algebraic NC analysis to differentiable delay bound

We restrict our description here to the two most commonly used curves in practice for industrial networks: the rate-latency service curve $\beta_{R,L}$ and the token-bucket arrival curve $\gamma_{r,B}$:

$$\beta_{R,L}(t) = R[t - L]^+, \forall t \geq 0 \tag{9}$$

$$\gamma_{r,B}(t) = B + r \cdot t, \forall t \geq 0 \tag{10}$$

with $[x]^+ = x$ if $x \geq 0$ and 0 otherwise. Note that our description is not limited to these curves as DiffNC can be extended to concave arrival curves and convex service curves.

Applying NC's (min,plus)-algebraic operations to the above curve types, the following lemma can be derived:

Lemma 1 (Closed-form expression of NC operations): With the assumption of using rate-latency service curves and token-bucket arrival curves, the NC operations listed in Section III-B have the following closed-form solutions:

$$\text{aggregation:} \quad \gamma_{r_1,B_1} + \gamma_{r_2,B_2} = \gamma_{r_1+r_2,B_1+B_2} \tag{11}$$

$$\text{convolution:} \quad \beta_{R_1,L_1} \otimes \beta_{R_2,L_2} = \beta_{\min(R_1,R_2),L_1+L_2} \tag{12}$$

$$\text{deconvolution:} \quad \gamma_{r,B} \oslash \beta_{R,L} = \gamma_{r,B+r\cdot L} \tag{13}$$

$$\text{left-over:} \quad \beta_{R,L} \ominus \gamma_{r,B} = \beta_{R-r,(B+R\cdot L)/(R-r)} \tag{14}$$

$$\text{delay bound:} \quad h(\gamma_{r,B}, \beta_{R,L}) = B/R + L \tag{15}$$

under the condition that $r < R$.

From Lemma 1, the following theorem is derived:

Theorem 2 (Differentiability of delay expression): With the assumption of using rate-latency service curves and token-bucket arrival curves, a NC end-to-end delay bound is differentiable w.r.t. the curves parameters.

Proof: Using the closed-form (min,plus) operations from Lemma 1, all NC operations use the following basic operators: addition, multiplication, division and min. For the min operator, we use the following partial derivates for $x \neq y$:

$$\frac{\partial \min(x,y)}{\partial x} = \begin{cases} 1, \text{ if } x < y \\ 0, \text{ if } x > y \end{cases} \qquad \frac{\partial \min(x,y)}{\partial y} = \begin{cases} 0, \text{ if } x < y \\ 1, \text{ if } x > y \end{cases}$$

All of the applied operators are then differentiable, proving Theorem 2. □

Partial derivates for the min operator used in the above proof are easily implemented using the Heaviside step function.

B. Generalized NC Model

As an exemplary optimization problem, we detail a formulation of the optimization of flow paths in a given network. A traditional NC network model has only one path per flow and all existing analyses are tailored to this basic model. I.e., our example problem would have to be tackled by first enumerating all combinations of paths and creating a network for each combination. Instead, we generalize the NC network model to hold all alternative paths of flows, alongside binary variables for mutual exclusion of alternatives.

Let $\mathcal{G} = (\mathcal{S}, \mathcal{E})$ be the directed server graph on which the end-to-end delay bounds of the set of flows $\mathcal{F}$ need to be optimized. I.e., we also depart from the assumption that there is only a single foi to analyze. Moreover, each flow f_i with arrival curve α_{f_i} may take multiple paths in $\mathcal{G}$ given its source and destination servers. We adopt here a path flow model, where a set of paths $\mathcal{P}_{f_i}$ are considered for each flow $f_i \in \mathcal{F}$ in the server graph. A fixed set of paths can easily be enumerated using traditional graph traversal algorithms.

For each flow f_i and each potential path $j \in \mathcal{P}_{f_i}$, we define $p_{f_{i,j}}$ as a binary variable representing the choice of path j for flow f_i. From this formulation, the following constraint forces to have a unique path for each flow:

$$\sum_{j \in \mathcal{P}_{f_i}} p_{f_{i,j}} = 1, \forall f_i \in \mathcal{F} \tag{16}$$

With these, we define so-called *virtual flows* along the different potential paths as illustrated in Figure 1. The input functions $A_{f_{i,j}}(t)$ of the virtual flows are set to 0 on the non-optimal paths, and are constrained by α_{f_i} on the optimal paths. For each virtual flow, Equation (2) is reformulated as:

$$\forall\, 0 \leq d \leq t \,:\, A_{f_{i,j}}(t) - A_{f_{i,j}}(t-d) \leq \alpha_{f_i}(d) \cdot p_{f_{i,j}} \tag{17}$$

By applying Lemma 1 to the virtual flow model, the token-bucket arrival curve of a virtual flow from Equation (17) is equivalent to:

$$\alpha_{f_i}(d) \cdot p_{f_{i,j}} = \gamma_{r_i \cdot p_{f_{i,j}}, B_i \cdot p_{f_{i,j}}}(d) \tag{18}$$

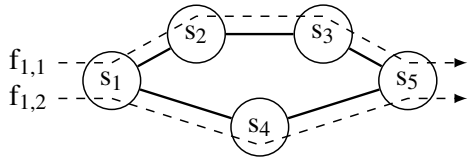

Figure 1: Illustration of virtual flow concept with one flow taking two potential paths in the server graph.

We essentially defined a mixed-integer nonlinear programming (MINLP), a combinatorial optimization problem with a number of potential solutions growing in $\mathcal{O}(|\mathcal{F}|^{|\mathcal{P}|})$.

Traditional NC analyses such as SFA or PMOO would not be able to capture the virtual flow model but instead include all flows in their resulting (min,plus)-algebraic NC term. I.e., their backtracking of flow dependencies succeeds but the transformation of these depencies to a term needs to be overly pessimistic. We extend the resulting (min,plus)-algebraic NC term with our binary variables, yet, unchanged traditional analyses cannot analyze this term anymore.

Note, that our formulation and subsequent applications to the optimization methods may also be reformulated as a link flow model. This alternate formulation avoids iterating over a fixed set of paths for each flow, but potentially requires more binary variables and their associated constraints.

C. Constrained nonlinear programming

Using DiffNC and Theorem 2, we extend NC to other applications, the main one being nonlinear optimization. As shown later in Section VI, NLP techniques based on gradient information – such as Newton's method – are well-known to outperform other NLP optimization techniques.

We show here how to model the network design problem as a differentiable NLP of the following form:

$$\min_{x \in \mathbb{R}^n} \quad f(x) \tag{19}$$

$$\text{s.t.} \quad g_l \leq g(x) \leq g_u \tag{20}$$

with $f()$ and $g()$ differentiable functions w.r.t. x, and g_l and g_u the upper and lower bounds for $g()$.

To make this problem solvable in polynomial time, we apply a commonly used technique known as relaxation, namely: the $p_{f_{i,j}}$ binary variables are relaxed as continuous variables on the interval $[0, 1]$. Following Theorem 2, the end-to-end delay bound expression of a virtual flow is then differentiable w.r.t. the $p_{f_{i,j}}$ variables. This relaxation technique transforms the MINLP into a continuous NLP, enabling the use of NLP methods based on gradient.

Using on the previous formulations, the following constrained nonlinear optimization problem is then defined:

$$\min_{p_{f_{i,j}}, \forall f_i \in \mathcal{F}, j \in \mathcal{P}_{f_i}} \quad \frac{1}{|\mathcal{F}|} \sum_{i,j} \textit{delay bound}(f_{i,j}) \cdot p_{f_{i,j}} \tag{21}$$

$$\text{s.t.} \quad 0 \leq p_{f_{i,j}} \leq 1, \forall f_i \in \mathcal{F}, j \in \mathcal{P}_{f_i} \tag{22}$$

$$\sum_{j \in \mathcal{P}_{f_i}} p_{f_{i,j}} = 1, \forall f_i \in \mathcal{F} \tag{23}$$

$$\sum_{i \in T(k)} r_i \cdot p_{f_{i,j}} \leq R_k, \forall k \in \mathcal{S} \tag{24}$$

with $T(k)$ the set of virtual flows traversing server k with service curve β_{R_k, L_k}, and *delay bound*$(f_{i,j})$ the end-to-end delay bound of the virtual flow $f_{i,j}$ computed with any of the classical algebraic NC analyses (i.e. SFA or PMOO). Equation (21) minimizes the average end-to-end delay bound in the networks. Following the previous theorems, Equations (21) to (24) are differentiable w.r.t the relaxed $p_{f_{i,j}}$ variables.

With our formulation, we also enable a wider range of constraints and objective functions. Constraints can be added such that a maximum delay requirement is satisfied for a given flow, saving us a subsequent check against the requirement:

$$\sum_{j \in \mathcal{P}_{f_i}} \textit{delay bound}(f_{i,j}) \cdot p_{f_{i,j}} \leq \textit{requirement} \tag{25}$$

This formulation is able to express complex objectives, enabling finer control over the type of solution which is required. A popular mathematical framework for describing hard or soft requirements on network performance (such as delay or bandwidth) is the concept of utility-based network optimization introduced in [4]. The objective function can be formulated with nonlinear utility functions U_i for the delay bounds:

$$\min_{p_{f_{i,j}}, \forall i,j} \sum_i U_i \left(\sum_j \textit{delay bound}(f_{i,j}) \cdot p_{f_{i,j}} \right) \tag{26}$$

with U_i a differentiable utility function mapping the delay bonds to a utility value in the interval $[0, 1]$.

Additionally, aspects such as the tail of the delay bound distribution can be minimized by defining the objective function:

$$\min_{p_{f_{i,j}}, \forall i,j} \max_i \left(\sum_j \textit{delay bound}(f_{i,j}) \cdot p_{f_{i,j}} \right) \tag{27}$$

As shown later in Section VII, the optimization formulation can be applied to any curve parameter, including the service curve parameters. This means that the optimization formulation can also be defined w.r.t. scheduler characteristics.

D. Automatic differentiation and optimization

We built with the previous theorems the mathematical foundations of DiffNC. We detail here how to put it into practice in order to compute efficient partial derivates of the end-to-end delay bounds w.r.t. the curves parameters.

While computer-assisted symbolic differentiation could be used for deriving closed-form expressions of the gradient, our initial numerical evaluations with SymPy [28] showed that this method had difficulties scaling to networks with 100+ flows.

To overcome this scalability issues, we selected AD. It is a family of techniques based on the calculus' chain rule for efficiently and accurately evaluating derivatives of numeric functions expressed as computer programs. This technique has gained a lot of popularity recently due its wide use in computing packages used for machine learning [29].

For our implementation, we selected CasADi [30], a software package enabling easy and efficient use of AD using a syntax similar to a CAS. Thanks to its support for reverse accumulation AD, the gradient of the objective function w.r.t. the curves' parameters can easily and efficiently be computed.

Combining CasADi with NC is straight-forward: We choose an analysis such as SFA or PMOO and execute its backtracking to derive a (min,plus)-algebraic NC term. Then we extend the term with the $p_{f_{i,j}}$ binary variables (see Section IV-B) and explicitly solve the operations according to Lemma 1. CasADi can now easily derive the gradient according to the underlying NC analysis. Combined with nonlinear optimization methods

using gradients – as detailed later in Section VI-A – our generalized network models can efficiently be optimized. Additionally, since most optimization methods require multiple evaluations of the objective function, CasADi also allows us to run the NC network analysis a single time and generate a so-called computation graph. This graph translates the delay expressions (i.e. the objective function) to a combination of basic operations (addition, multiplication, etc.). The objective function can then be evaluated multiple times, without requiring to run the NC-specific parts again.

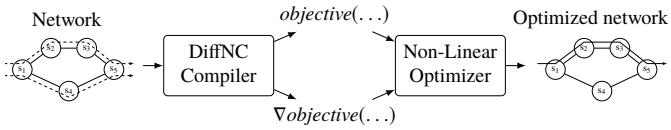

Figure 2: Illustration of the system for optimizing networks.

This process is illustrated in Figure 2. A network to-be-optimized with potential flow paths is used as input of our framework. Based on the end-to-end delay bounds calculations, the computation tree of the objective function and its gradient, and the constraint functions and its gradients, are then generated and compiled. The compiled formulas are then used as input to a nonlinear optimizer supporting the generalized NLP presented in Equations (19) and (20).

V. Non-algebraic NC alternative

DiffNC is not the first attempt at combining optimization techniques with NC. An LP formulation of a NC model was proposed in [3]. It converts the equations introduced in Section III to linear constraints, under the assumption that curves are piecewise linear functions, either concave or convex. Flows are backtracked to derive constraints capturing, among others, their mutual impact. Complexity grows exponentially when computing a flow's tight delay bound and a heuristic called unique linear program (ULP) was proposed. The ULP was shown to have only limited scalability [20], yet it is our only hope for a non-algebraic NC competitor. We complement the ULP to include our $p_{f_i,j}$ variables and to optimize for multiple flows. The resulting formulation is able to find configurations, yet preliminary evaluation already shows that it scales poorly.

The ULP is based around two classes of variables. Time variables $t_h \in \mathbb{R}^+$ represent departure or arrival time of bits of data of the flows at the different servers of the network. Function variables $A^{s_k}_{f_i}(t_h) \in \mathbb{R}^+$ represent the departure and arrival processes of the data of flows at the different servers of the network, i.e. the arrival and departure functions introduced in Section III as $A(t)$ and $A'(t)$.

Based on these variables, the ULP translates arrival curves from Equation (2) as linear constraints:

$$A^{s_k}_{f_i}(t_{h+1}) - A^{s_k}_{f_i}(t_h) \leq \alpha_i(t_{h+1} - t_h), \forall s_k, f_i \tag{28}$$

and similarly service curves from Equation (8) as:

$$\sum_{f_i} \left(A^{s_k}_{f_i}(t_{h+1}) - A^{s_k}_{f_i}(t_h) \right) \geq \beta_k(t_{h+1} - t_h), \forall s_k \tag{29}$$

Additional constraints representing, e.g. causality are also added. We refer to [3] for a full formulation.

We extend here this formulation to take into account different paths for the flows. For each flow i and each potential path j, we define the variables $A^{s_k}_{f_{i,j}}(t_h)$ as the departure and arrival processes of the data of the virtual flows along the path j. The variables $A^{s_k}_{f_{i,j}}(t_h)$ are constrained as in the original formulation from [3] as if they were normal flows. Following Equation (17), the following constraints are added:

$$A^{s_k}_{f_{i,j}}(t_h) \leq M \cdot p_{f_{i,j}}, \forall s_k, f_{i,j}, t_h \tag{30}$$

with M a large constant chosen such that $\alpha_i(t_{h+1} - t_h) \leq M, \forall t_{h+1}, t_h$ in the LP formulation. Using the big-M method, Equation (30) achieves the same effect as Equation (17): the $A^{s_k}_{f_{i,j}}(t_h)$ are constrained to 0 on the paths where $p_{f_{i,j}} = 0$ – i.e. removing their impact on the delay calculation of the other flows – and leaving them unconstrained when $p_{f_{i,j}} = 1$.

While this ULP formulation of the optimal routing problem is attractive, it suffers from two important drawbacks: difficulty for expressing delay constraints and optimization goals, and poor scalability. The first drawback of this approach is that some requirements regarding the optimization problem are not straightforward to translate into the ULP. This drawback mainly stems from the fact that the delay bound itself is calculated by maximizing an expression in the ULP. This leads to difficulty at implementing an objective function which would minimize average delay bounds. Similarly, adding a constraint regarding a maximum delay requirement as in Equation (25) is not straightforward: the objective function maximizes the delay bound, but such a constraint would result in an underestimation of the delay bound in some cases.

Secondly, as noted in [3, 31] and subsequently numerically illustrated in [20], the ULP is only tractable on relatively small networks due to its exponentially growing number of constraints. To illustrate this point, we evaluated our modified ULP including the $p_{f_i,j}$ variables and the constraints from Equation (30) on a set of randomly generated networks. Details about the networks are explained later in Section VI-C. For the objective function, we maximize the sum of delay bounds. We extended the ULP implementation from NCorg DNC v2.6.2 [32] for our evaluation. Figure 3 illustrates the time to find a solution with a time limit of 1 hour using IBM's CPLEX 20.1.0 on an Intel Xeon Gold 5120 at 2.2 GHz.

As expected, the solve time grows exponentially, exceeding the one hour time limit even on small networks. This result highlights why an alternative solution for optimizing networks is necessary for larger networks. As a further motivating comparison, Figure 3 also illustrates the optimization time on the same networks with our contributed approach.

VI. Numerical evaluation

We evaluate in this section our approach on a wide range of networks. We illustrate its scalability and compare it against other optimization methods.

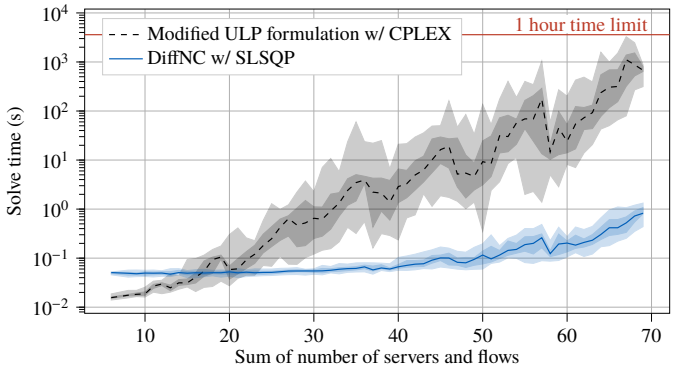

Figure 3: Solve time of the ULP formulation against network size using CPLEX. The curves represent respectively the 10, 25, 50, 75 and 90 percentiles.

A. Implementation of DiffNC

Part of our DiffNC implementation was already described in Section IV-D. We detail here the nonlinear optimization part using gradient-based constrained optimization methods with open-source implementations. Our implementation directly calls the respective C++ API either from the NLOpt library [33], or the respective implementations from [34, 35].

1) NLP solvers: First, we selected sequential least squares quadratic programming (SLSQP) based on the implementation from [36, 33]. Secondly, we used the method of moving asymptotes (MMA) [37] and the conservative convex separable approximation (CCSA) method [38] in conjunction with an augmented Lagrangian method [39, 40] in order to include the equality constraints from Equation (16). For both algorithms, the implementation from [33] is used. Finally, we also evaluated Interior Point OPTimizer (IPOPT) [34, 41], which is a primal-dual interior point method. For all theses methods, integer relaxation was used. The final solution is then converted back to integer and verified against the constraints.

2) MINLP solver: We also evaluated the Basic Open-source Nonlinear Mixed INteger programming (BONMIN) [35], a MINLP solver which does not explicitly require integer relaxation using IPOPT as sub-solver for the NLP.

Note that these are local optimization methods, each requiring a starting point. For our evaluation and metrics, a single randomly generated starting point was used.

B. Other heuristics

To benchmark our approach against potential competitors, the following other optimization methods were selected. They include both naïve and greedy approaches, as well as other optimization techniques often used for solving constrained combinatorial problems. Except for the randomized search, a maximum of 500 evaluations of the objective function has been defined for the heuristics described here.

1) Randomized search: In this greedy approach, a random number of combinations of paths are chosen and evaluated. The combination leading to the best objective is kept. In the figures, this method is labeled as Random(M=m), with m the number of random combinations evaluated.

2) Hop-count shortest path: For this approach, the path minimizing the number of hops for each flow is selected. This approach does not use other information about the network and is equivalent to a traditional Dijkstra shortest-path algorithm.

3) Delay-based shortest path: This approach is similar to the previous one, except that we partially take into account the arrival and service curves in the network. For each flow and each potential path, we perform an end-to-end SFA NC delay analysis as if the flow was the only flow in the network [42]. Without crosstraffic, a virtual flow's delay is then $h(\gamma_{r,B}, \beta_{R_j,L_j}) = L_j + \frac{B}{R_j}$ with R_j the minimum rate on its path $j \in \mathcal{P}_i$ and L_j the sum of server latencies. The path leading to the minimum end-to-end delay is selected.

4) Meta-heuristic algorithms: Various meta-heuristic optimization algorithms based on nature have been proposed in the literature such as evolutionary algorithms. For our evaluation, we selected the following algorithms from the pygmo library [43]: artificial bee colony [44], particle swarm optimization [45], covariance matrix adaptation-evolution strategy [46], and exponential evolution strategy [47].

5) Non-gradient-based nonlinear optimization: Finally, we also included compass search [48], an iterative direct search method for global optimization.

C. Evaluated networks

To numerically evaluate our approach, we randomly generated a set of evaluation networks. First, a random amount of servers was generated, connected in a directed graph. Each server has a rate-latency service curve, with rate and latency parameters randomly sampled from a uniform distribution. A random amount of source-destination pairs was then generated for flows, each with a token-bucket arrival curve, with rate and burst parameters randomly sampled from a uniform distribution. For each pair, a set of virtual flows were generated according to the available paths in the directed graph.

For each network, the minimization of the average end-to-end delay bound of the flows is used as objective function, computed using SFA under the assumption of arbitrary multiplexing. In order to ease the integration with our framework, we implemented our own NC analysis tool in C++.

Overall, our dataset contains topologies with up to 1000 flows, matching the number of flows found in some industrial settings [5, 6, 7]. Table I contains statistics about the dataset.

Table I: Statistics about the generated dataset

Number of	Min	Mean	Median	Max
Servers	8	17.08	16	31
Flows	5	170.67	164	1001
Virtual flows	9	355.22	343	1884
Path combinations	$10^{1.08}$	$10^{46.04}$	$10^{44.10}$	$10^{229.08}$

Additionally for the numerical evaluation performed in Section V and Figure 3, a dataset containing smaller networks was also generated using the same approach. Table II contains relevant statistics about this additional dataset.

Table II: Statistics about the networks used for the evaluations in Section V and Figure 3

Number of	Min	Mean	Median	Max
Servers	3	8.68	8	18
Flows	3	9.70	9	21
Virtual flows	4	18.62	17	45
Path combinations	$10^{0.30}$	$10^{2.07}$	$10^{1.81}$	$10^{5.52}$

D. Reduction of delay bounds

We evaluate here the solution of each optimization method presented in Section VI-B on our evaluation dataset. We use here Equation (21) as objective function, i.e. we minimize the average end-to-end delay bound in the networks, an objective function found in many other related works. Such an objective cannot be expressed in the ULP described in Section V.

We first compare then optimization methods using the result of the hop-based shortest path approach as a baseline. We use the relative gap of the objective function our metric, namely:

$$RelGapShortestPath_{\text{method}} = \frac{objective_{\text{method}}}{objective_{\text{shortest path}}} - 1 \quad (31)$$

Since we aim at minimizing the delays, a negative value of the relative gap means that the evaluated optimization method achieved better results than simply using shortest path.

Results are presented in Figure 4. DiffNC with SLSQP is able to achieve the best results compared to all the other heuristics evaluated here. Overall, it achieved a reduction of 27.5 % of the average delay bounds. Compared to the evolution-based meta-heuristics, all gradient-based optimization methods based on DiffNC achieve much better results. Interestingly, the delay-based shortest path approach is able to surpass the evolution-based meta-heuristics, showing that a simple heuristic using domain knowledge about model and analysis used in the optimization problem can be effective.

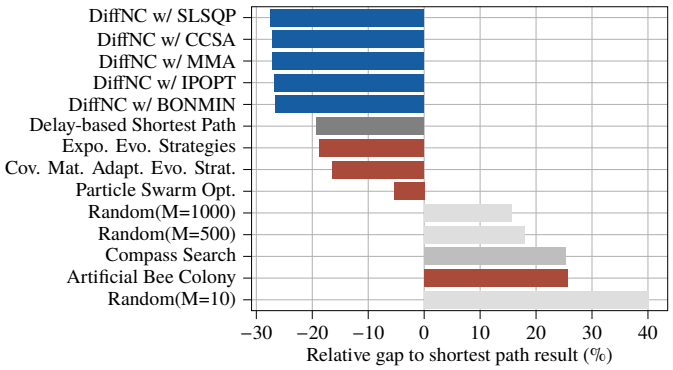

Figure 4: Average relative gap to the result of shortest path. Negative values mean optimizations outperform shortest path.

Given the large number of path combinations in some networks (larger than 10^{229} in some cases), the optimal network configurations are not known and cannot be computed in reasonable time by simply enumerating the combinations. To address this, we use the best result which was obtained by any evaluated method as a baseline. We use the relative gap of the objective function to the best objective as metric:

$$RelGapBest_{\text{method}} = \frac{objective_{\text{method}}}{objective_{\text{best}}} - 1 \quad (32)$$

Additionally, as our experiments showed that DiffNC with SLSQP outperformed all the other methods, we decided to also run it multiple times with different randomly generated starting points since SLSQP is a local optimization method. This enables us to sample additional combinations, potentially closing the gap to the optimal solution.

Results are presented in Figure 5. With an average relative gap of 0.14 %, DiffNC with SLSQP with a single run achieves the best results compared to all the other heuristics, outperforming them by at least one order of magnitude. By running it multiple times, the relative gap was 0 %, meaning it always outperformed all the other methods. The observations made from Figure 4 for the other methods also apply for Figure 5.

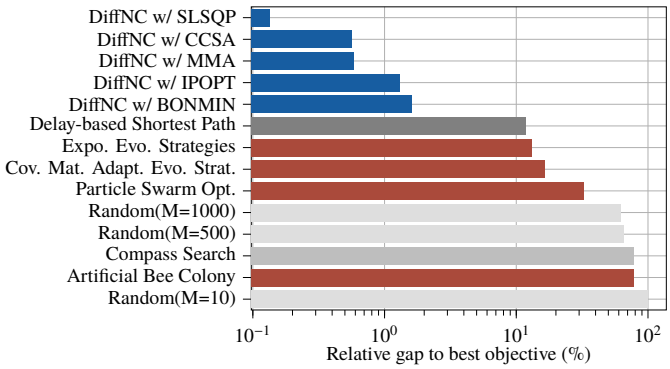

Figure 5: Average relative gap to the best objective. A value close to zero indicates a solution close to the best one.

A detailed view of the distribution of the relative gap is presented in Figure 6. DiffNC with SLSQP consistently outperforms the other methods by at least an order of magnitude.

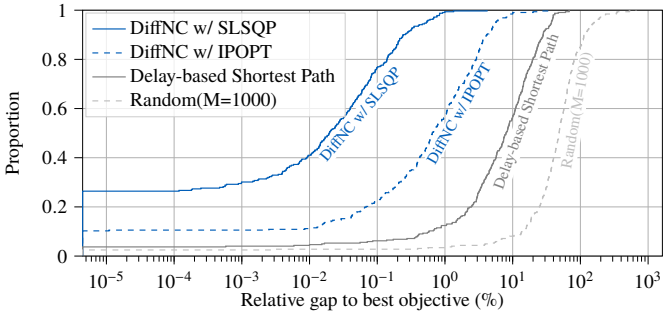

Figure 6: CDF of the relative gap to the best objective. A value close to zero indicates a solution close to the best one.

E. Optimality gap

We evaluate the gap of DiffNC to the optimum found by bruteforcing, restricted to networks from Table II where it terminates within 1 h. Table III shows the results: SLSQP

without restarts found the optimum in 85.3 % of networks. Given the limit of 500 iteration steps, we can restart the optimization with another initial point to find a better result. In this case, SLSQP reached even 99.53 % optimal solutions.

Table III: Optimality against a bruteforce approach

Method	Optimum found	Rel. gap to bruteforce	Avg. exec. time
Bruteforce	-	-	123.05 s
DiffNC w/ SLSQP w/o restarts	85.30 %	0.17 %	0.05 s
DiffNC w/ SLSQP w/ restarts	99.53 %	7.1×10^{-4} %	0.17 s

F. Impact of computation tree construction

We evaluate here the impact of building a computation tree of the delays performed with our implementation of DiffNC on our dataset from Table I. As for the evaluation presented in Section V, execution times are measured here on an Intel Xeon Gold 5120 at 2.2 GHz.

Our system targets optimization methods where multiple evaluations of the delay formulas are required. To avoid recomputing the full NC term for each execution of the delay formula, we take advantage of CasADi's computation tree and its compilation to assembly (x86-64 in our case). Figure 7 illustrates the total time taken for executing the NC analysis, compiling the computed delay formula, and evaluating it N times. For small values of N, directly executing the analysis is actually faster since building and compiling the computation tree takes some time. Yet, when the delay formula is required to be executed more than about 50 times, a gain of more than one order of magnitude in execution time can be achieved on networks with less than 100 virtual flows. On larger networks, this gain is less apparent due to a larger cost of generating and compiling the computation tree.

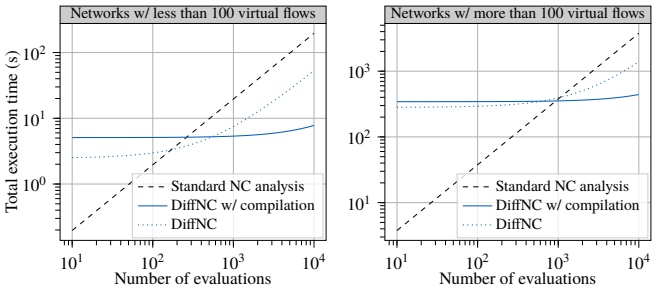

Figure 7: Evaluation of the total execution time.

Figure 8 illustrates the individual execution time of the end-to-end delay formulas against all the individual NC analyses without building a computation tree or the compilation step. Compared to executing the NC analyses directly, a gain of about two orders of magnitude can be achieved using the compiled computation tree. Figures 7 and 8 illustrate that our implementation is able to take advantage of the compiled computation tree to scale it to large networks.

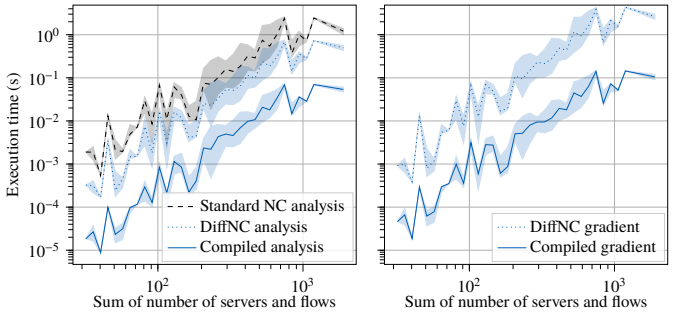

Figure 8: Execution time of the delay formulas with standard floats, with DiffNC and execution time of the gradients.

G. Execution time

Following our discussion on the ways to optimize the computation speed of DiffNC, we compare here the execution time of optimization part of DiffNC against the other heuristics. The durations presented here do not include the compilation time discussed in Section VI-F. Results are presented in Figure 9. Due to the limit of 500 evaluations of the objective function, most of the algorithms exhibit here similar execution times.

Overall, DiffNC with CCSA is the slowest of all the methods, with a median execution time of 40.6 s. While it is the method with the largest computation time, it is still on par with the other meta-heuristics, which have similar execution time. We note that by using DiffNC with BONMIN, we are able to reduce the execution by half, with a median execution time of 1.21 s, making it faster than other DiffNC-methods. As shown earlier in Figure 5, DiffNC with BONMIN is still able to achieve a good relative gap w.r.t. the objective compared with the other methods having similar execution times.

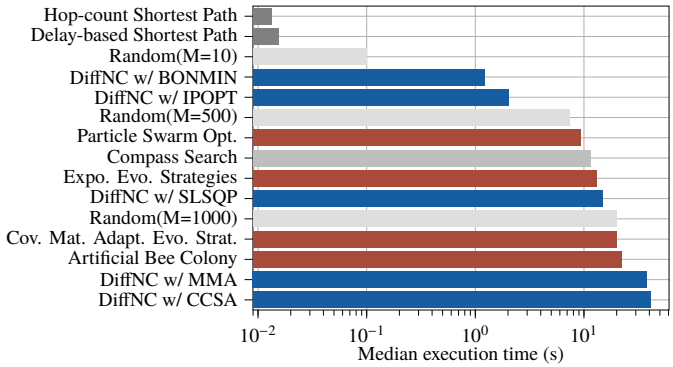

Figure 9: Median execution time to find the optimal solution once the objective function has been compiled.

Overall, our evaluations show that DiffNC is an efficient method for optimizing networks under delay bound constraints, outperforming all the other optimization methods evaluated here, and at a reasonable computational cost. This also applies to large networks with up to 1000 flows, illustrating that this method scales to realistic industrial networks.

VII. Discussion

A. Application to TSN

We present here an application of DiffNC to the optimization of TSN networks, a set of standards bringing determinism to Ethernet networks. While TSN encompasses a large set of options, we take a simplified view of TSN's mechanisms.

We assume the following: flows are classified into eight priority classes. The first priority class is time scheduled, while the other priority classes are constrained using a rate limiter. The optimization task is then to optimize: (i) the paths, (ii) their priorities, (iii) and the time schedule for the network.

We show here that the approach presented in Section IV-B can be combined simultaneously with these optimization tasks.

1) Priority optimization: Priority-based scheduling can easily be modeled using NC's left-over service curve principle [2]. To find the optimal priority for a flow, we reuse the virtual flow principle and apply it to priorities. Namely, a virtual flow is created for each potential priority class and assigned the priority of the given class, as illustrated in Figure 10.

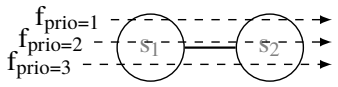

Figure 10: Illustration of virtual flow concept for priority.

For each flow f_i, each potential path j, and each priority k, we define $p_{f_{i,j,k}}$ as a binary variable representing the choice of path j and priority k for flow f_i. The arrival curve of each virtual flow is constrained by $p_{f_{i,j,k}}$ as in Equation (17). The sum of $p_{f_{i,j,k}}$ variable is equal to 1 to enforce that only one virtual flow is selected, as in Equation (16).

2) Schedule optimization: TSN includes a scheduling mechanism standardized in IEEE 802.1Qbv, based on a time schedule. As shown in various works [49, 50], this may be modeled using NC to compute end-to-end delay bounds. For this section, we simplify the TSN service as a TDMA schedule modeled using the following service curve:

$$\beta_{\text{TDMA}}(t) = R \cdot \max\left(\left\lfloor \frac{t}{c} \right\rfloor s, t - \left\lceil \frac{t}{c} \right\rceil (c-s)\right) \tag{33}$$

with c the cycle length, and s the sleep amount. It can be demonstrated that with some loss of tightness, this service curve can then be simplified as a rate-latency service curve:

$$\beta(t) = R\frac{s}{c}\left[t - (c-s)\right]^+ \tag{34}$$

The s and c parameters are free parameters constrained such that $s \leq c$. This formulation is then used in conjunction with DiffNC, and the end-to-end delay formula is differentiable according to the s and c parameters.

B. More complex curves

As presented in Lemma 1, our approach hinges upon a closed-form formulation of the (min,plus) algebra as a combination of differentiable operations. While we restricted this paper to token-bucket and rate-latency curves, it can be extended to more general arrival and service curves.

In most cases, the more complex curves can be upper or lower bounded by token-bucket or rate-latency curves, as it was shown in Section VII-A and [51]. This impacts the tightness of the model, but simplifies the later operations. In case of curves with a fixed number of linear segments, closed-form solutions can also be derived, as in Lemma 1.

C. Extension to other network design problems

Other parameters from the NC analysis can also be optimized. Various NC models of packet schedulers were already proposed in the literature such as Weighted Fair Queueing [2] or Deficit Round Robin [52]. These models are compatible with DiffNC's differentiability, enabling to optimize their parameters (i.e. weights). Additionally, FIFO multiplexing includes a θ parameter [53] which can be freely chosen. Its optimal value may also be derived using DiffNC.

D. Use for machine learning-based NC

As noted in Section II, ML recently attracted attention in the NC community [23, 24, 25, 26]. These approaches use neural networks (NNs) for speeding up resource intensive analyses by multiple orders of magnitude. While they use NC as a black box, DiffNC can be plugged in these models. This enables back-propagation from the bounds up to the weights of the NNs, which has been shown to speed-up the training [54, 55].

VIII. Conclusion

We introduce in this paper Differential Network Calculus (DiffNC), an extension of Network Calculus, by showing that the (min,plus)-algebric terms derived by NC are differentiable. The term bounding a flow's end-to-end delay can already be differentiated w.r.t. to curve parameters or flow priorities. Yet, our approach also enables for network design and synthesis.

We investigate the optimization of flows paths, a task known to be difficult due to its combinatorial nature. An extension of NC models to include alternative flows paths allows to differentiate w.r.t. these. We show that DiffNC with variable relaxation is able to reformulate the optimization problem as a constrained nonlinear optimization problem that can be optimized using gradient-based methods. Our numerical evaluation shows that DiffNC with sequential least squares quadratic programming can reduce average delay bounds by 27.5 % compared to shortest path routing. Moreover, a comparison against other optimization methods for combinatorial and nonlinear optimization, demonstrates that DiffNC can outperform global search methods and evolution-based methods.

Compared to a linear program formulation from the literature which is not tractable beyond small networks, DiffNC is able to scale to network sizes found in industrial settings. Our approach also enables more complex objective functions, allowing greater flexibility for real-world applications.

The authors have provided public access to their data at `https://github.com/fabgeyer/dataset-infocom2022`.

References

[1] F. Geyer and G. Carle, "Network engineering for real-time networks: comparison of automotive and aeronautic industries approaches," *IEEE Commun. Mag.*, vol. 54, no. 2, pp. 106–112, 2016.

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

[3] A. Bouillard, L. Jouhet, and É. Thierry, "Tight performance bounds in the worst-case analysis of feed-forward networks," in *Proc. of IEEE INFOCOM*, 2010.

[4] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, pp. 237–252, 1998.

[5] M. Boyer, N. Navet, and M. Fumey, "Experimental assessment of timing verification techniques for AFDX," in *Proc. ERTS*, 2012.

[6] D. Tămaş-Selicean, P. Pop, and W. Steiner, "Design optimization of TTEthernet-based distributed real-time systems," *Real-Time Syst.*, vol. 51, no. 1, pp. 1–35, Jan. 2015.

[7] R. Belliardi, J. Dorr, T. Enzinger, F. Essler, J. Farkas, M. Hantel, M. Riegel, M.-P. Stanica, G. Steindl, R. Wamßer, K. Weber, and S. A. Zuponcic, "Use cases IEC/IEEE 60802 – v1.3," 2018.

[8] A. Bouillard, B. Gaujal, S. Lagrange, and E. Thierry, "Optimal routing for end-to-end guarantees using network calculus," *Performance Evaluation*, vol. 65, no. 11-12, pp. 883–906, 2008.

[9] B. Cattelan and S. Bondorf, "Iterative design space exploration for networks requiring performance guarantees," in *Proc. of IEEE/AIAA DASC*, 2017.

[10] B. D. de Dinechin, Y. Durand, D. van Amstel, and A. Ghiti, "Guaranteed services of the NoC of a manycore processor," in *Proc. of the NoCArc Workshop*, 2014.

[11] E. Schweissguth, D. Timmermann, H. Parzyjegla, P. Danielis, and G. Muhl, "ILP-based routing and scheduling of multicast realtime traffic in time-sensitive networks," in *Proc. of IEEE RTCSA*, 2020.

[12] S. M. Laursen, P. Pop, and W. Steiner, "Routing optimization of AVB streams in TSN networks," *ACM SIGBED Review*, 2016.

[13] S. Vastag, "Modeling quantitative requirements in SLAs with network calculus," in *Proc. of ValueTools*, 2011.

[14] P. Buchholz and S. Vastag, "Toward an analytical method for SLA validation," *Software & Systems Modeling*, vol. 17, no. 2, 2018.

[15] J. Liebeherr, "Duality of the max-plus and min-plus network calculus," *Found. Trends. Network.*, vol. 11, no. 3-4, pp. 139–282, 2017.

[16] M. Boyer and P. Roux, "Embedding network calculus and event stream theory in a common model," in *Proc. of IEEE ETFA*, 2016.

[17] K. Lampka, S. Perathoner, and L. Thiele, "Analytic real-time analysis and timed automata: A hybrid method for analyzing embedded real-time systems," in *Proc. of ACM EMSOFT*, 2009.

[18] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Proc. of ISCAS*, 2000.

[19] A. Bouillard and G. Stea, "Exact worst-case delay in FIFO-multiplexing feed-forward networks," *IEEE/ACM Trans. Netw.*, 2015.

[20] S. Bondorf, P. Nikolaus, and J. B. Schmitt, "Quality and cost of deterministic network calculus – design and evaluation of an accurate and fast analysis," *Proc. ACM Meas. Anal. Comput. Syst. (POMACS)*, vol. 1, no. 1, pp. 16:1–16:34, 2017.

[21] A. Bouillard, "Trade-off between accuracy and tractability of network calculus in FIFO networks," *Performance Evaluation*, vol. 153, p. 102250, 2022.

[22] D.-K. Dang and A. Mifdaoui, "Timing analysis of TDMA-based networks using network calculus and integer linear programming," in *Proc. of IEEE MASCOTS*, 2014.

[23] F. Geyer and S. Bondorf, "DeepTMA: Predicting effective contention models for network calculus using graph neural networks," in *Proc. of IEEE INFOCOM*, 2019.

[24] ——, "Graph-based deep learning for fast and tight network calculus analyses," *IEEE Trans. Netw. Sci. Eng.*, 2020.

[25] F. Geyer, A. Scheffler, and S. Bondorf, "Tightening Network Calculus Delay Bounds by Predicting Flow Prolongations in the FIFO Analysis," in *Proc. of IEEE RTAS*, 2021.

[26] T. L. Mai and N. Navet, "Improvements to deep-learning-based feasibility prediction of switched Ethernet network configurations," in *Proc. of RTNS*, 2021.

[27] S. Bondorf and J. B. Schmitt, "Calculating accurate end-to-end delay bounds–you better know your cross-traffic," in *Proc. of ValueTools*, 2015.

[28] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, "SymPy: symbolic computing in Python," *PeerJ Computer Science*, 2017.

[29] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *Journal of Machine Learning Research*, vol. 18, no. 153, pp. 1–43, 2018.

[30] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, 2019.

[31] A. Bouillard, "Algorithms and efficiency of network calculus," Habilitation à Diriger des Recherches, École Normale Supérieure (Paris), 2014.

[32] S. Bondorf and J. B. Schmitt, "The DiscoDNC v2 – a comprehensive tool for deterministic network calculus," in *Proc. of ValueTools*, 2014.

[33] S. G. Johnson, "The NLopt nonlinear-optimization package – version 2.7.0," 2020.

[34] A. Wächter, "An interior point algorithm for large-scale nonlinear optimization with applications in process engineering," Ph.D. dissertation, Carnegie Mellon University, 2002.

[35] P. Bonami, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya *et al.*, "An algorithmic framework for convex mixed integer nonlinear programs," *Discrete Optimization*, vol. 5, no. 2, pp. 186–204, 2008.

[36] D. Kraft, "A software package for sequential quadratic programming," DFVLR, Institut für Dynamik der Flugsysteme, Germany, Tech. Rep. DFVLR-FB 88-28, 1988.

[37] K. Svanberg, "The method of moving asymptotes—a new method for structural optimization," *International journal for numerical methods in engineering*, vol. 24, no. 2, pp. 359–373, 1987.

[38] ——, "A class of globally convergent optimization methods based on conservative convex separable approximations," *SIAM journal on optimization*, vol. 12, no. 2, pp. 555–573, 2002.

[39] M. R. Hestenes, "Multiplier and gradient methods," *Journal of optimization theory and applications*, vol. 4, no. 5, pp. 303–320, 1969.

[40] M. J. Powell, "A method for nonlinear constraints in minimization problems," *Optimization*, pp. 283–298, 1969.

[41] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[42] S. Bondorf and J. B. Schmitt, "Boosting sensor network calculus by thoroughly bounding cross-traffic," in *Proc. of IEEE INFOCOM*, 2015.

[43] F. Biscani and D. Izzo, "A parallel global multiobjective framework for optimization: pagmo," *Journal of Open Source Software*, 2020.

[44] D. Karaboğa, "An idea based on honey bee swarm for numerical optimization," Erciyes University, Tech. Rep. TR06, 2005.

[45] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of International Conference on Neural Networks*, 1995.

[46] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evolutionary computation*, 2003.

[47] T. Glasmachers, T. Schaul, S. Yi, D. Wierstra, and J. Schmidhuber, "Exponential natural evolution strategies," in *Proc. of the Conference on Genetic and Evolutionary Computation*, 2010.

[48] T. G. Kolda, R. M. Lewis, and V. Torczon, "Optimization by direct search: New perspectives on some classical and modern methods," *SIAM review*, vol. 45, no. 3, pp. 385–482, 2003.

[49] L. Zhao, P. Pop, and S. S. Craciunas, "Worst-case latency analysis for IEEE 802.1Qbv Time Sensitive Networks using network calculus," *IEEE Access*, vol. 6, pp. 41 803–41 815, 2018.

[50] L. Zhao, P. Pop, Z. Zheng, H. Daigmorte, and M. Boyer, "Latency analysis of multiple classes of AVB traffic in TSN with standard credit behavior using network calculus," *IEEE Trans. Ind. Electron.*, vol. 68, no. 10, 2021.

[51] K. Lampka, S. Bondorf, J. B. Schmitt, N. Guan, and W. Yi, "Generalized finitary real-time calculus," in *Proc. of IEEE INFOCOM*, 2017.

[52] M. Boyer, G. Stea, and W. M. Sofack, "Deficit round robin with network calculus," in *Proc. of ValueTools*, 2012.

[53] R. L. Cruz, "SCED+: Efficient management of quality of service guarantees," in *Proc. of IEEE INFOCOM*, 1998.

[54] T. Rocktäschel and S. Riedel, "End-to-end differentiable proving," in *Advances in Neural Information Processing Systems*, 2017.

[55] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," in *Advances in Neural Information Processing Systems*, 2018.