Efficient Gradient-based Network Calculus for Scalable Synthesis of Network Configurations

Fabien Geyer
Airbus Central Research & Technology
Munich, Germany

Steffen Bondorf
Faculty of Computer Science
Ruhr University Bochum, Germany

***Abstract*—Standards aiming for a bounded networking delay are being developed and deployed at an unprecedented pace, in particular in industrial settings such as avionics. Simultaneously, modeling and analysis techniques are being extended to derive formally verified delay bounds for these networks. Network Calculus (NC) has seen much attention, including a first evolutionary step from an analysis method to a synthesis method called Differential Network Calculus (DiffNC). That is, while classical NC requires a fully specified network for model derivation and analysis, DiffNC can find delay-bound-optimizing settings for open configuration parameters. Yet, synthesis of parameter values increases the computational demand drastically such that computational efficiency becomes imperative. We implemented an entirely new tool chain for DiffNC, allowing for parallelization and offering a gradient-based nonlinear optimization algorithm more suited for that task. We demonstrate scalability by optimizing two orthogonal configuration options as well as application to the network of the Airbus A350. In numerical evaluations we show improved delay bounds by up to factor two and runtime reduction by one order of magnitude.**

***Index Terms*—Network Calculus, Gradient-based Optimization, Automatic Differentiation, Parameter Synthesis**

I. Introduction

The use of ever more detailed mathematical models for formal derivation of performance bounds has become common practice in various industries [1]. Developing networking solutions with real-time requirements is usually based on standards like IEEE Time Sensitive Networking (TSN) or Avionics Full-Duplex Switched Ethernet (AFDX) that then offer a multitude of configuration parameters to the network engineers. This triggers two of the major challenges to the formal analysis of models: they are inherently combinatorial and additionally oftentimes nonlinear. Put simple, computing delay bounds in polynomial time is hard – let alone with optimizing a network design with open configuration parameters in mind. Therefore, previous attempts for doing so have often been limited to small network sizes with only a limited amount of interdependent configuration options being explored.

In this paper, we focus on improving a recently presented approach for modeling, optimizing, and synthesizing networks under hard end-to-end delay constraints: Differential Network Calculus (DiffNC). We aim at scalability to networks of realistic sizes and illustrate our contribution by finding flow paths and flow priorities.

While the underlying Network Calculus (NC) is commonly used in some industries to formally validate delay requirements, it is rarely used for synthesis or as a design tool. Existing analyses have been designed to analyze already completed network designs, making them only suitable for design space exploration that enumerates and ranks different designs. DiffNC extends NC to a method for synthesis by leveraging gradient-based nonlinear programming (NLP) and automatic differentiation (AD).

While previous version of DiffNC [2, 3] were based on freely available off-the-shelf software, we implemented our own AD tool for this work, highly specialized for NC's (min,plus) operations and delay bound calculation. As shown later in Section IV-F, our implementation enables for better scalability on large networks and to parallelize various parts of the code. This stems from the fact that most available AD tools are targeting relatively small computation graphs (i.e., series of mathematical operations described as a directed graph) with arithmetically intense operations such as large matrix multiplications. This specialization makes these tools poorly scalable for NC operations, as millions of (min,plus) operations are required to compute delay bounds in industrial use-cases. Our tool also directly uses the (min,plus) operations, enabling for better scalability compared to a conversion to basic mathematical operations.

Conceptually, a network to-be-optimized with alternative flow paths is used as input to our framework. Based on the end-to-end delay bounds calculations, the computation graph of the objective function and its gradient, and the constraint functions and its gradients, are then generated and compiled for our DiffNC evaluation. They are then used as input to a nonlinear optimizer supporting the created NLP.

To solve the NLP, we present the Frank-Wolfe algorithm, an efficient NLP based on the well-known gradient descent algorithm. We show that it can be used to find a good solution in terms of optimality in a short amount of time. In our numerical evaluation, we demonstrate that we are able to optimize the AFDX network used in the Airbus A350 in a matter of seconds, with better optimality than previous works. We also illustrate that our approach is scalable to real networks with more than 1000 flows.

This paper is organized as follows: Section II presents the background of DiffNC and Section III provides details on our efficiency improvements to differentiation and optimization in NC. We numerically evaluate these improvements in Section IV. Section V concludes the paper.

2325-1301/25/$31.00 ©2025 IEEE
DOI 10.1109/RTCSA66114.2025.00029

II. Deterministic Network Calculus

A. Classic DNC modeling and analysis

Network Calculus (NC) modeling is built around two concepts [4, 5]: *a)* the network of deterministic queueing locations crossed by constrained data flows, and *b)* resource modeling with cumulative functions in interval time. Based on a fully specified model, a traditional NC analysis derives an upper bound on a single analyzed flow's end-to-end delay.

The behavior at queueing locations can be defined by a wide range of scheduling policies, ranging from single first-in first-out (FIFO) queues, over round robin schedulers (e.g., WRR, DRR) to a combination of priority queues, shaping, time-triggered scheduling etc. as, for example, defined by TSN. To separate service provision for schedulers with static configurations, the respective vertex is split into the defined number of priority levels or round robin classes. We call these vertices the servers and the resulting graph the server graph $\mathcal{G} = (\mathcal{S}, \mathcal{E})$ where each server $s \in \mathcal{S}$ forwards queued data. The guaranteed service is expressed by non-negative, non-decreasing functions

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

NC uses univariate functions that give deterministic bounds on either data arrivals or forwarding in time intervals of duration d, called curves in interval time. Curves are in $\mathcal{F}_0$, an extension of $\mathcal{F}_0^+$ by $\forall t < 0 : f(t) = 0$. called service curves. For later input to the Separate Flow Analysis (SFA)[1], we additionally require the notion of strictness:

Definition 1 (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.

The NC literature provides a rich set of results to derive the service curves for different schedulers' service classes, e.g., TSN [7, 8].

Definition 2 (Data Flow): A data flow (short flow) crosses the graph $\mathcal{G}$ from a source server to a given number of destination servers, both in $\mathcal{S}$. Its unicast or multicast flow path is a connected subgraphs of $\mathcal{G}$.

For deterministic analysis, a constraint on the data sent by a flow needs to be enforced and assumed to be known at its source. NC models such constraints as follows:

Definition 3 (Arrival Curve): Let data flow f send data into a network. Assume the cumulative amount of data sent by f up until (absolute) time t is described by the function $A \in \mathcal{F}_0^+$. Then, a function $\alpha \in \mathcal{F}_0$ is an arrival curve (in interval time) for f iff

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

That is, in no interval of any duration d, the data flow f will send more data than specified by $\alpha(d)$.

[1]An NC analysis derives a delay bounding term from the server graph, expressed in (min,plus)-algebraic operations. Similarly, Real-time Calculus (RTC) [6] defines that term by modeling a network of components. Our work is based on this term, see Equation (10), and can be easily adapted to RTC.

Regarding the shape of the above curves, we restrict our models to those commonly used in practice for industrial networks: the rate-latency service curve $\beta_{R,L}$ and the token-bucket arrival curve $\gamma_{r,B}$.

$$\beta_{R,L}(t) = R[t-L]^+, \forall t \ge 0 \quad (3)$$

$$\gamma_{r,B}(t) = [B + r \cdot t]_{\{t>0\}}, \forall t \ge 0 \quad (4)$$

with R, L, r, and B in $\mathbb{R}^+$, and $[x]^+ = x$ if $x \ge 0$ and 0 otherwise, and $[x]_{\{condition\}} = x$ if the *condition* is fulfilled and 0 otherwise. If necessary, we index the variables with the index of the flow whose shape they define.

Analysis of interaction between these bounds on resource demand and supply is captured by a set of (min,plus)-algebraic operations. A comprehensive presentation can be found in [4, 5]. For brevity, our presentation is restricted to the closed-form expressions for the above curve shapes.

Lemma 1 (Closed-form expression of NC operations):

$$\text{aggregation:} \quad \gamma_{r_1,B_1} + \gamma_{r_2,B_2} = \gamma_{r_1+r_2,B_1+B_2} \quad (5)$$

$$\text{concatenation:} \quad \beta_{R_1,L_1} \otimes \beta_{R_2,L_2} = \beta_{\min(R_1,R_2),L_1+L_2} \quad (6)$$

$$\text{output bounding:} \quad \gamma_{r,B} \oslash \beta_{R,L} = \gamma_{r,B+r\cdot L} \quad (7)$$

$$\text{left-over:} \quad \beta_{R,L} \ominus \gamma_{r,B} = \beta_{R-r,(B+R\cdot L)/(R-r)} \quad (8)$$

$$\text{delay bound:} \quad h(\gamma_{r,B}, \beta_{R,L}) = B/R + L \quad (9)$$

under the condition $r \le R$ that guarantees bounded queues.

In this article, we will apply the SFA [4] as the set of rules to derive the (min,plus)-algebraic, delay-bounding term from the server graph. For example, bounding the worst-case end-to-end delay of flow 1 (f_1) in the server graph shown in Figure 1 under the assumption of arbitrary multiplexing [9] would yield

$$h(\ \alpha_{f_1},\ (\beta_{s_1} \ominus (\alpha_{f_2} + \alpha_{f_3})) \otimes \beta_{s_2} \otimes \beta_{s_3} \otimes (\beta_{s_5} \ominus (\alpha_{f_2} \oslash ((\beta_{s_1} \ominus (\alpha_{f_1} + \alpha_{f_2})) \otimes \beta_{s_4}))))$$

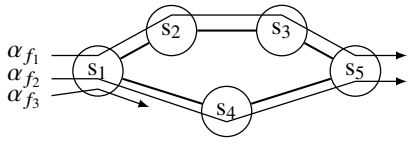

Figure 1: Sample server graph consisting of five servers and three flows.

B. Generalized modeling and analysis for constrained NLP

The steps towards computing a delay bound as described in Section II-A cater to an analysis that requires a fully specified model such as SFA. I.e., in case there are design alternatives, each of these needs to be fully specified and analyzed independently. A related topic are NC analyses that explore alternative orders of (min,plus)-operations internally. There, exhaustive enumeration may be feasible [10] but is often prohibitive [11]. For either case, it was shown to be possible to design heuristics that vastly increase efficiency at little cost in terms of loss of delay bound accuracy [12, 13].

To solve this challenge differently, DiffNC defines the following constrained nonlinear optimization problem that minimizes the average delay bound:

$$\min_{p_{f_{i,j}}, \forall f_i \in \mathcal{F}, j \in \mathcal{P}_{f_i}} \quad \frac{1}{|\mathcal{F}|} \sum_{i,j} \textit{delay bound}(f_{i,j}) \cdot p_{f_{i,j}} \tag{10}$$

$$\text{s.t.} \quad 0 \leq p_{f_{i,j}} \leq 1, \forall f_i \in \mathcal{F}, j \in \mathcal{P}_{f_i} \tag{11}$$

$$\sum_{j \in \mathcal{P}_{f_i}} p_{f_{i,j}} = 1, \forall f_i \in \mathcal{F} \tag{12}$$

$$\sum_{i \in T(k)} r_i \cdot p_{f_{i,j}} \leq R_k, \forall k \in \mathcal{S} \tag{13}$$

with $T(k)$ the set of alternatives considered for a flow to traverse server k, strict service curve β_{R_k,L_k}, and *delay bound*$(f_{i,j})$ the end-to-end delay bound of the flow alternative $f_{i,j}$ computed with any of the classical algebraic NC analyses (e.g., SFA). Note, that we thus also generalized the analysis to consider the delay bounds of multiple flows $f_{i,j}$ simultaneously whereas the classical analyses can only analyze a single flow of interest at a time. Due to the operations in Lemma 1, Equation (10) is a non-convex objective function.

To make this problem solvable in polynomial time, DiffNC applies a commonly used technique known as relaxation, namely: the $p_{f_{i,j}}$ binary variables are relaxed as continuous variables on the interval $[0, 1]$. 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 mixed-integer nonlinear programming (MINLP) into a continuous NLP, enabling the use of NLP methods based on gradient information.

Given the nature of the NC analyses such as SFA, the solver will yet provide us with a binary solution to the relaxed $p_{f_{i,j}}$ variables as we will exemplarily illustrate in Section III-A1.

III. Differentiation and Optimization in NC

Efficient optimization is key for scalable synthesis of configuration parameters with the DiffNC formulation presented above – i.e., to find a design alternative in a potentially large design space, subject to NC delay bounds and at very low cost in terms of computational costs as well as loss of delay bound accuracy (deviation from the optimum). Naturally, the addition of multiple alternatives like flow paths and priorities raises challenges in analysis scalability. Application of gradient-based NLP algorithms were found to serve this purpose and we provide a novel, highly efficient differentiation and optimization tool chain in this section. In this section, we provide model and tooling while Section IV evaluates scalability.

A. Generalized, parameterized NC model

We generalize the server graph model of NC to a more comprehensive, parameterized one where newly added parameters define design space alternatives. We simply allow each variable in the model (e.g., those defining an arrival or strict service curve) to remain open and each part of the NC model (e.g., entire curves) to be accompanied by an open parameter. The added parameters can, e.g., be binary to decide whether the parameterized part of the model is to be considered; continuous parameters can be added for weighing parameterized parts. Combinations of parameters can also be restricted to express certain mutually exclusive design alternatives. Without further restrictions on the parameters or their combinations, finding a parameter setting may become a MINLP. In this paper, we exemplarily consider two instances of the generalized, parameterized NC model.

1) Alternative flow paths: Let $\mathcal{G} = (\mathcal{S}, \mathcal{E})$ be the server graph of the analyzed communication network and let $\mathcal{F}$ be the set of flows crossing $\mathcal{G}$. We adopt here a path flow model, where each flow $f_i \in \mathcal{F}$ is associated with multiple alternative paths $\mathcal{P}_{f_i}$. We amend each potential path $j \in \mathcal{P}_{f_i}$, or rather the data sent over j by f_i, with a relaxed binary variable $p_{f_{i,j}}$ such that Equation (2) becomes:

$$\forall\, 0 \leq d \leq t \,:\, A_{f_{i,j}}(t) - A_{f_{i,j}}(t-d) \leq \alpha_{f_i}(d) \cdot p_{f_{i,j}} \tag{14}$$

That is, we add the information of all potential path of flows to a server graph, including the respective flow's arrival curve that is equal for all paths. We call the alternatives along the different potential paths *virtual flows*. Yet, only a single path should be taken, which we achieve by

- Equation (12) capping to at most one alternative in total,
- the inherent disadvantage of any non-binary decomposition of the relaxed $p_{f_{i,j}}$ variables in NC,
- returning the closest binary solution.

The disadvantage of any non-binary decomposition is inherited from the NC analyses. Assume a flow f_i with arrival curve α_{f_i} is divided into two sub-flows by variables $0 < p_{f_{i,1}} < 1$ and $p_{f_{i,2}} = 1 - p_{f_{i,1}}$. These sub-flows must share at least two servers, the source and the destination server. Any NC analysis regards the two sub-flows as independent and assumes they interfere with each other – cf. flows f_2 and f_3 interfering with f_1 in Figure 1. As a result, the SFA would then include the following terms at the source server: $\beta_s \ominus (\alpha_{f_i} \cdot p_{f_{i,1}})$ and $\beta_s \ominus (\alpha_{f_i} \cdot p_{f_{i,2}})$. I.e., the NLP's objective function may bound the real system behavior with a model including self-interference. This effectively discourages non-binary decomposition, yet, as NLP algorithms may not deliver optimal results, non-binary decomposition may still be proposed by them. In our large and complex test networks, we observed that almost always one single alternative dominated, leading to $p_{f_{i,j}}$ values very close to 1 and 0. Our tool returns the closest binary solution and in case a tie breaker is required, we choose randomly. Virtual flow arrival curves would thus be set to 0 on the presumably non-optimal paths, and remain constrained by α_{f_i} on the optimal paths. See Figure 2 for an illustration of the model.

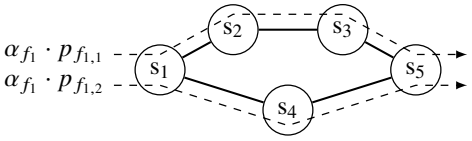

Figure 2: Illustration of virtual flow concept with one flow taking two potential paths in the server graph.

Due to our formulation, servers may be overloaded, leading to invalid network configurations. Hence the following stability constraint is required for each server s in the server graph:

$$\sum_{f \in \mathcal{F}_s} rate_f \leq rate_s \tag{15}$$

with $\mathcal{F}_s$ the set of flows traversing server s.

As expected, we essentially defined a MINLP, a combinatorial optimization problem with a number of potential solutions growing in $\mathcal{O}(|\mathcal{F}|^{|\mathcal{P}|})$ with $\mathcal{P} = \bigcup_{f_i \in \mathcal{F}} \mathcal{P}_{f_i}$.

2) Flow priority assignment: We can use the virtual flow concept to define the search space for (network-wide) priority assignment. Namely, a flow f_i is extended to a set of virtual flows, one for each priority class. For each virtual flow of f_i with its potential path j and potential priority class 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 then amended by $p_{f_{i,j,k}}$, analog to Equation (14), which are relaxed for optimization and then mapped back to a binary solution. Figure 3 illustrates this model.

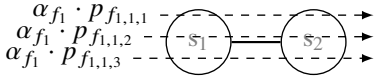

Figure 3: Virtual flow concept for priority assignment.

B. Automatic Differentiation

The previous theorems build the mathematical foundations of DiffNC. Conceptually, extending them to a DiffNC toolchain is straight-forward. Figure 4 illustrates the steps. For efficient and scalable execution of the toolchain, we focus on optimizing the individual parts following the NC theorems.

The initial presentation of DiffNC [2] as well as follow-up work [3] relied on off-the-shelf software, leaving considerable room for improvement. Tools for automatic differentiation (AD) take a computational graph as input, i.e., a directed graph representing the operations to execute as well as their dependencies. However, the common assumption of these tools is that graphs are rather small and operations rather complex (e.g., large matrix multiplications). For an NC analysis, the exact opposite is true. The server graph may very be large, and that extends to the delay-bounding term an analysis such as SFA derives from it. I.e., we will have a large and complex computational graph with millions of (min,plus) operations representing the objective function of Equation (10). Yet, each of the (min,plus) operations is relatively simple and simple to differentiate, see Lemma 1.

In [2], CasADi [14] was used and the authors of [3] extended JAutoDiff [15]. For this paper, we implemented our own AD tool in Go. It is tailored to NC by directly using (min,plus) operations and the delay bound calculation instead of basic mathematical operations. This enables for better scalability on large networks and to parallelize various parts of the code.

C. The Frank-Wolfe algorithm

Nonlinear optimization has been shown to outperform linear optimization formulations derived from previous work in NC [2] as well as search-based algorithms [3]. We detail here gradient-based constrained optimization. The Frank-Wolfe algorithm [16, 17] – also known as conditional gradient method – is our main solution for solving the NLP. To optimize an NLP with objective function f, the algorithm executes the following loop. Given a solution x_k at iteration k:

- *Step 1:* Find the solution s_k to the linearized version of the NLP using the gradient information (i.e., $s_k^T \triangledown f(x_k)$),
- *Step 2:* Update $x_{k+1} \leftarrow x_k + \delta(s_k - x_k)$.

The choice of δ determines the so-called step size of the Frank-Wolfe algorithm.

While this algorithm was designed to solve convex NLP, it was shown to also perform well on non-convex problems in practice by choosing $\delta = 1/\sqrt{k+1}$ [17]. We evaluate the gap of DiffNC to the optimum found by exhaustive enumeration, restricted to networks from [2][2] where the analysis terminates within 1 h. Table I shows the results: sequential least squares quadratic programming (SLSQP) [18] and Frank-Wolfe were able to respectively find the optimum in 89.1 % and 83.0 % of networks. The relative gap to the optimum is of 0.1 % and 0.3 %, outperforming all the other methods. The method of moving asymptotes (MMA) [19] in conjunction with an augmented Lagrangian method [20, 21] is shown as it can include the equality constraints from Equation (12). Yet, it is outperformed by the other heuristics, even by a random approach that simply evaluates 500 combinations chosen uniformly at random.

Section IV-F extends this insight by an evaluation of the time elapsed to compute results.

Table I: Number of networks where the optimal solution from exhaustive enumeration was found and average relative gap to the optimal solution.

Method	Opt. found	Rel. gap to opt
DiffNC w/ SLSQP	89.1 %	0.1 %
DiffNC w/ Frank-Wolfe	83.0 %	0.3 %
Random	62.4 %	3.9 %
DiffNC w/ MMA	52.5 %	29.8 %

D. Parallelization

We evaluate in this section the impact of parallelization of the DiffNC toolchain. For our implementation, we use different strategies for parallelization. First, we parallelize the different delay bound computations both during the preparation of the (min,plus) terms and during the execution of DiffNC. These computations are independent of each other and our objective function from Equation (10) requires the computation of the delay bounds of all flows in the network. Secondly, we also parallelized part of the computations during

[2]Online https://github.com/fabgeyer/dataset-infocom2022

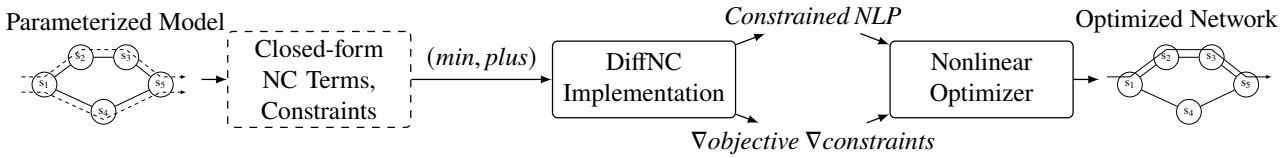

Figure 4: Illustration of the conceptual DiffNC proceedings. First, the required (min,plus) operations are prepared as (min,plus) operations for our implementation. They are then used during the iterative optimization process to avoid unnecessarily computing them at each optimization iteration. The gradient is computed using a backward pass on the (min,plus) operations.

the preparation of the (min,plus) terms of the SFA network analysis.

The impact of the parallelization is highlighted in Figure 5, where the delay bound calculations of all flows from the AFDX network from the Airbus A350 were performed. We notice that, as we increase the number of cores used for the computations, the total execution time of the analysis is reduced. Overall, a gain of more than one order of magnitude was possible thanks to parallelization.

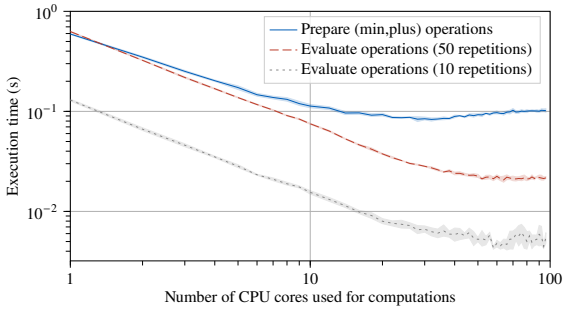

Figure 5: Impact of parallelization on the DiffNC execution times for delay bounding in the Airbus A350 AFDX network network. Measured on AMD EPYC 7702P.

IV. 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.

A. Competing nonlinear optimization algorithms for DiffNC

For our evaluation, we benchmark against SLSQP as it was shown in [2] to perform the best in terms of optimality as well as MMA. For all these algorithms, the implementation from [22] is used.

Relaxation was used and the final solution is then converted back to integer and verified against the constraints. 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. A maximum of 500 evaluations of the objective function has been defined for the heuristics described here.

1) Randomized search: In this greedy approach, 500 random combinations of paths are chosen and evaluated. The combination leading to the best objective is presented. In the figures, this method is labeled as Random.

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) Minimum-delay 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. The minimum delay bound that an NC end-to-end delay analysis can compute for a flow is in the absence of crosstraffic impact, e.g., if the flow had highest priority among all flows. In that case, the end-to-end delay bound is $h(\gamma_{r,B}, \beta_{R_j,L_j}) = L_j + \frac{B}{R_j}$ with R_j the minimum rate on its path $j \in \mathcal{P}_i$ and L_j the sum of server latencies. We use this as a proxy metric and select the path with minimum end-to-end delay in hypothetical absence of crosstraffic.

4) Non-gradient based methods: We also evaluate the Nelder-Mead [23] and Subplex [24] algorithms, both direct search methods based on the simplex algorithm. Neither algorithm makes use of the gradient but Subplex was found to perform exceptionally well in the DiffNC application in [3].

Our original work [2] already evaluated other heuristics based on meta-heuristics or evolutionary algorithms. They were omitted here since our previous evaluation showed that they underperformed compared to DiffNC.

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 strict 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.

The goal of the optimization is to find the best path and the best priority (high or low, network-wide) for a given flow. For each network, the minimization of the average end-to-end delay bound of the flows is used as objective function, computed using multicast SFA under the assumption of arbitrary multiplexing [9].

Our dataset contains topologies with up to 1000 flows, matching the number of flows found in industrial settings [25, 26, 27]. Table II contains statistics about the dataset.

Table II: 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	$10^{1.08}$	$10^{46.04}$	$10^{44.10}$	$10^{229.08}$
Path + priority comb.	$10^{2.58}$	$10^{97.41}$	$10^{94.28}$	$10^{530.41}$

Finally, we also used the AFDX network from the Airbus A350 as a representative industrial network for our evaluations in Sections IV-E and IV-F. This network contains approximately 1100 multicast flows (called virtual links) with an average of 8 destinations per multicast flow. In this industrial network, multicast paths are fixed such that we focus only on optimizing the (network-wide) priority of the flows.

D. Reduction of delay bounds

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

$$RelGapShortestPath_{\text{method}} = \frac{objective_{\text{method}}}{objective_{\text{shortest path}}} - 1 \quad (16)$$

Since we aim at minimizing delay bounds, 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 6. DiffNC with Frank-Wolfe is able to achieve the best results compared to all the other heuristics evaluated here, closely followed by SLSQP. Overall, Frank-Wolfe achieved a reduction of 39.25 % of the average delay bounds. Compared to the non-gradient-based algorithms, all gradient-based optimization methods based on DiffNC achieve much better results. Interestingly, the delay-based shortest path approach is able to surpass the non-gradient-based optimization methods, showing that a simple heuristic using domain knowledge about the model and analysis used in the optimization problem can be somewhat effective.

Given the large number of path and priority level combinations in some networks (larger than 10^{530} 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, called here virtual best. We use the relative gap of the objective function to the best objective as metric:

$$RelGapBest_{\text{method}} = \frac{objective_{\text{method}}}{objective_{\text{virtual best}}} - 1 \quad (17)$$

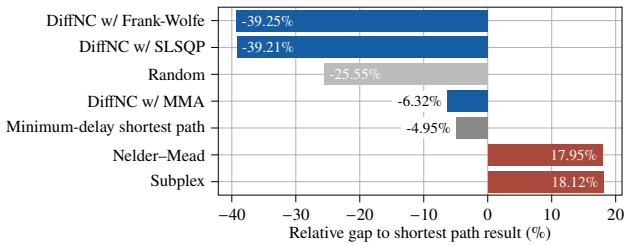

Figure 6: Average relative gap to the result of shortest path. Negative values mean optimizations outperform shortest path.

Results are presented in Figure 7. With an average relative gap of 0.25 %, DiffNC with Frank-Wolfe achieves the best results compared to all the other heuristics. DiffNC with SLSQP closely follows it with an average relative gap of 0.56 %. Both methods outperform all the other heuristics by at least one order of magnitude. Most observations made from Figure 6 for the other methods also apply for Figure 7. There is only one noteworthy exception: the NLP algorithms not using gradient information (Nelder-Mead and Subplex) are not beaten by the hop-count shortest path w.r.t. the metric "relative gap to the virtually best method".

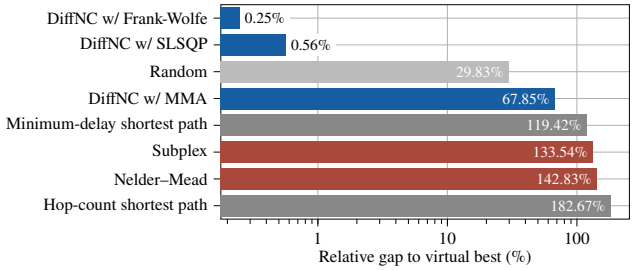

Figure 7: Average relative gap to the best objective. A value close to zero indicates a solution close to the best one.

E. Application to an industrial network

We evaluate in this section our approach on the AFDX network from the Airbus A350. We use the average reduction in delay bound compared to using only one priority level as our metric for evaluating DiffNC, namely:

$$\frac{objective_{n\ priorities}}{objective_{one\ priority}} - 1 \quad (18)$$

Results are presented in Figure 8. DiffNC with Frank-Wolfe is able to outperform both DiffNC with MMA and with SLSQP. This result confirms the conclusions from Section IV-D where a similar behavior was observed.

F. Execution time

Following our discussion on the ways to optimize the computation speed of DiffNC, we compare here the execution time of the optimization part of DiffNC against the other heuristics. Results are presented in Figure 9. Due to the limit of 500

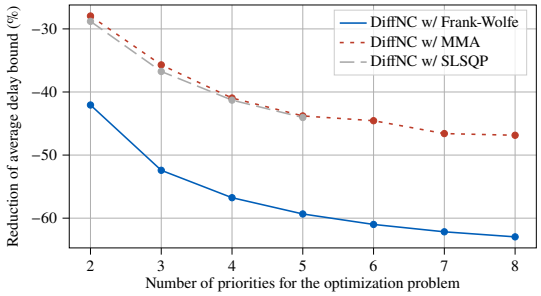

Figure 8: Reduction of delay bounds on AFDX network compared to using only one priority in the network. Results for DiffNC with SLSQP with more than 5 priorities are omitted here since they take more than 10 h to compute.

evaluations of the objective function, most of the algorithms exhibit here similar execution times. The evaluations presented here were performed on an AMD EPYC 7702P with 128 cores.

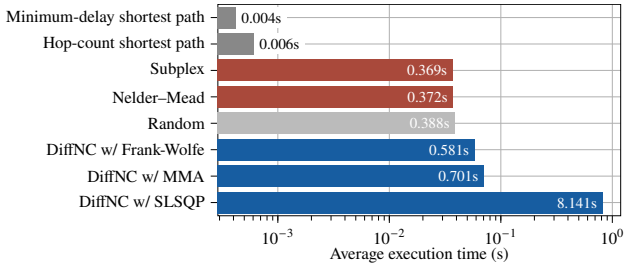

Figure 9: Average execution time to find the optimal solution.

One outlier is SLSQP which is one order of magnitude slower than the other methods. This is due to the fact that its execution time grows quadratically with the number of variables of the optimization problem. To illustrate this issue, we used DiffNC on the AFDX network, where we incrementally increase the number of virtual links in the network. Results are presented in Figure 10. DiffNC with SLSQP is almost 3 orders of magnitude slower than DiffNC with Frank-Wolfe on the full network, showing its poor scalability.

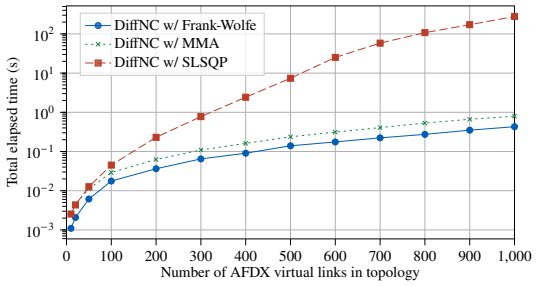

Figure 10: Scalability of DiffNC on the AFDX network when optimizing for two network-wide priority levels.

To further illustrate the scalability of DiffNC with Frank-Wolfe, we optimized the AFDX network with an increasing number of (network-wide) priorities. Results are presented in Figure 11. The execution time grows linearly with the number of priorities. Compared to DiffNC with SLSQP, DiffNC with Frank-Wolfe is three orders of magnitude faster.

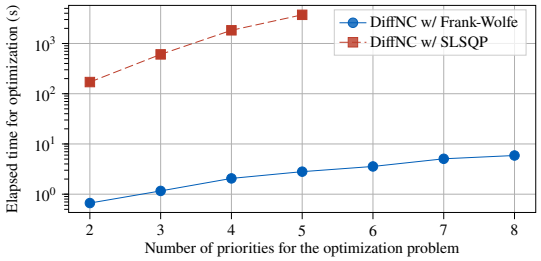

Figure 11: Execution time required for optimizing the AFDX network using DiffNC w/ Frank-Wolfe and SLSQP. Results for SLSQP with more than 5 priorities are omitted here since they take more than 10 h to compute.

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

V. Conclusion

In this paper, we improved upon Differential Network Calculus (DiffNC), an extension of Network Calculus (NC) based on the differentiability of the (min,plus)-algebraic terms derived by NC. The term bounding the end-to-end delay of a flow can already be differentiated w.r.t. to curve parameters or flow priorities. Paired with fast optimization methods, we show how our approach is able to scale to industrial networks for network design and synthesis.

We investigate the optimization of flow paths and priority assignment, a task known to be difficult due to its combinatorial nature. An extension of NC models to include alternative flow 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 Frank-Wolfe can reduce the average delay bounds by 39.2 % compared to shortest path routing.

We demonstrate that DiffNC with Frank-Wolfe is able to optimize the Avionics Full-Duplex Switched Ethernet network from the Airbus A350 in a matter of seconds, outperforming DiffNC with sequential least squares quadratic programming by several orders of magnitude. Furthermore, a comparison with other optimization methods for combinatorial and nonlinear optimization shows that DiffNC can outperform global search methods.

Future work directions for DiffNC could be the synthesis of scheduler-defining parameters as well as the extension to more complex curve shapes by extending Lemma 1.

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] F. Geyer and S. Bondorf, "Network synthesis under delay constraints: The power of network calculus differentiability," in *Proc. of IEEE INFOCOM*, May 2022.

[3] L. Herll and S. Bondorf, "Non-linear programming for the network calculus analysis of FIFO feedforward networks," in *the 16th ACM/SPEC International Conference on Performance Engineering (ICPE 2025)*, May 2025. [Online]. Available: https://dnet.cs.rub.de/publications/2025-HB-1.pdf

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

[5] A. Bouillard, M. Boyer, and E. Le Corronc, *Deterministic Network Calculus: From Theory to Practical Implementation*. John Wiley & Sons, Ltd, 2018.

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

[7] 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.

[8] 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.

[9] S. Bondorf and F. Geyer, "Generalizing network calculus analysis to derive performance guarantees for multicast flows," in *the 10th International Conference on Performance Evaluation Methodologies and Tools (ValueTools 2016)*, Oct. 2016.

[10] 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.

[11] S. Bondorf, "Better bounds by worse assumptions – improving network calculus accuracy by adding pessimism to the network model," in *Proc. of IEEE ICC*, 2017.

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

[13] F. Geyer, A. Scheffler, and S. Bondorf, "Network calculus with flow prolongation – a feedforward FIFO analysis enabled by ML," *IEEE Trans. Comput.*, vol. 72, no. 1, pp. 97–110, 2023.

[14] 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.

[15] JAutoDiff. An automatic differentiation library (pure java). [Online]. Available: https://github.com/uniker9/JAutoDiff

[16] M. Frank and P. Wolfe, "An algorithm for quadratic programming," *Naval Research Logistics Quarterly*, vol. 3, no. 1-2, pp. 95–110, 1956.

[17] G. Braun, A. Carderera, C. W. Combettes, H. Hassani, A. Karbasi, A. Mokhtari, and S. Pokutta, "Conditional gradient methods," Nov. 2022. [Online]. Available: https://conditional-gradients.org/

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

[19] 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.

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

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

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

[23] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[24] T. H. Rowan, "Functional stability analysis of numerical algorithms," Ph.D. dissertation, 1990.

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

[26] 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.

[27] 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.