The Case for a Network Calculus Heuristic: Using Insights from Data for Tighter Bounds

Fabien Geyer
Technical University of Munich (TUM)
D-85748 Garching b. München, Germany
Email: fgeyer@net.in.tum.de

Georg Carle
Technical University of Munich (TUM)
D-85748 Garching b. München, Germany
Email: carle@net.in.tum.de

***Abstract*—Deterministic network calculus offers a framework for providing guaranteed bounds on end-to-end delay and buffer usage in computer networks. Various network analysis methods have been proposed in order to reduce the impact of burstiness or multiplexing and provide tight performance bounds. Yet, the choice of which analysis method to use given a network to analyze is not straightforward as it has been shown in the literature that corner cases exist leading to poor tightness. We propose in this paper to take a new look at this question using insights from data and confirm that there is no clear winner when deciding which method to use. Based on those first results, we make the case for a network calculus heuristic in order to predict the bounds produced by a given network analysis method. Our main contribution is a heuristic based on graph-based deep learning, which is able to directly process networks of servers and flows. Via a numerical evaluation, we show that our proposed heuristic is able to accurately predict which analysis method will produce the tightest delay bound. We also demonstrate that the computational cost of our heuristic makes it of practical use, with average runtimes one or two order of magnitude faster than traditional analysis methods.**

I. Introduction

Performance guarantees are an essential part of network architecture and design in real-time networks such as safety critical systems [1]. In the case of large Ethernet networks, deterministic network calculus (DNC) [2] has been successfully used as a mathematical framework for validating and guaranteeing end-to-end delay requirements and buffer sizes. An important aspect of such framework is to achieve good tightness, namely minimizing the gap due to the pessimism of a method and the real worst-case. Various network analysis methods have been investigated in the literature to address this, by either focusing on specific effects such as multiplexing [3] or using methods based on optimization [4]. Yet, no clear guideline has been proposed with regards to choosing the appropriate method given a network to analyze.

To address this question, we propose in this paper to take a new look at this problem from the perspective of data. In a first step, we confirm previous results on corner cases of network analyses by evaluating them on randomly generated feed-forward networks. Based on those findings, we propose a heuristic for network calculus analyses using a neural network able to process graphs in order to predict the end-to-end latency bound of a given analysis method. Our heuristic is based on a transformation from the feed-forward server graph and the flows crossing it to a general graph which can then be analyzed using recent techniques from neural networks focused on graph analysis.

We demonstrate via a numerical evaluation that our trained neural network is able to predict end-to-end latency bounds with a relative median error of 2.5 %. While the predictions of the neural network cannot be directly used for giving performance guarantees, we show that such heuristic can be used for deciding which network analysis method to apply given a network and a flow of interest. Compared to a strategy of using only one analysis method, using our heuristic as a selection of which method to use leads to a relative reduction of the end-to-end delay bound of 12.79 % in average. Finally, we also evaluate the runtime of our heuristic and show that it is one order of magnitude faster than a standard network analysis method, highlighting its usability in practice.

The rest of this paper is organized as follows. We first give an overview of network calculus in Section II and investigate the state of the different network analysis methods using numerical evaluations. We present in Section III our heuristic based on graph-based deep learning. In Section IV, we numerically evaluate our approach and show an application of our heuristic. Related research studies are presented in Section V. Finally, Section VI concludes our work.

II. Deterministic network calculus

We present in this section a brief overview over deterministic network calculus and a numerical evaluation of its different network analysis methods. In DNC, a flow corresponds to unidirectional communications between a source and a destination, modeled as a function of its cumulative arrival of data. In order to compute bounds on flows, we are interested in the functions $A(t)$ corresponding to the data arriving in a given server s at time t, and $A'(t)$ the amount of data processed by the server at time t. Using this formalism, the following delay definition can then be derived:

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

$$D(t) = \inf\{\tau \geq 0 \mid A(t) \leq A'(t+\tau)\}$$

Instead of directly working with A, DNC makes use of the concept of arrival curves, which is a function bounding the maximal arrivals of a flow:

Definition 2 (Arrival curve): Given a flow with input A, a function α is an arrival curve for A iff

$$A(t) - A(s) \leq \alpha(t-s), \forall t, s, 0 \leq s \leq t$$

Definition 3 (Service curve): If the service provided by a server s for a given input A results in an output A', then s offers a service curve β iff

$$A'(t) \geq \inf_{0 \leq s \leq t} \{A(t-s) + \beta(s)\}, \forall t$$

A. (min, +) Algebra

DNC was formalized as a $(\min, +)$-algebraic framework in [5, 2], enabling an easier description of operations on flow and server descriptions. The $(\min, +)$ convolution and deconvolution of two functions f, g are defined as:

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

$$\text{Deconvolution: } (f \oslash g)(t) = \sup_{s \geq 0} \{f(t+s) - g(s)\}$$

Using those $(\min, +)$ operations, one can rewrite the previous definitions as $A' \geq A \otimes \beta$ and $A \otimes \alpha \geq A$. Moreover, $(\min, +)$ convolution allows DNC to concatenate the service of consecutive servers $\langle 1, \ldots, n \rangle$ into a single service curve.

B. Network analysis methods

We review here the most common network analysis methods from DNC. We refer to [6] for a more in-depth review of the different methods proposed in the literature. We call the analyzed flow *flow of interest*, abbreviated here *foi*. The foi's path defines the sequence of servers that defines its end-to-end delay. Different methods have been proposed in the literature for bounding this end-to-end delay.

1) Total Flow Analysis (TFA) [2]: The TFA first computes per-server delay bounds. Each one holds for the sum of all the traffic arriving to a server. The flow's end-to-end delay bound is derived by summing up the individual server delay bounds on its path.

2) Separated Flow Analysis (SFA) [2]: The SFA is a direct application of other theorems: first compute the left-over service of each server on the foi's path, then concatenate them and finally derive the end-to-end delay bound. Deriving the end-to-end delay bound using only one service curve will consider the burst term of the foi only once, a property called Pay Burst Only Once (PBOO).

3) Pay Multiplexing Only Once (PMOO) [3]: The PMOO analysis first convolves the tandem of servers before subtracting the cross-traffic. Using this order, the bursts of the cross-traffic appear only a single time compared to the SFA analysis where the bursts are included at each server. Therefore, multiplexing with cross-traffic is only paid for once.

4) Arrival bounding methods: For more involved feed-forward networks, a procedure to combine tandem analyses with a network analysis have been proposed. [7] established two basic steps of the analysis: *1)* cross-traffic arrival bounding and *2)* flow of interest performance bounding. For the cross-traffic arrival bounding, the flows interfering with the foi are backtracked to their sources to derives the dependencies between the foi and its cross-flows in a recursive fashion. The PBOO and PMOO properties can then also be applied on the cross-traffic as shown in [8, 7].

C. Numerical comparison between methods

From the description of the different methods previously presented, the most promising method for producing a tight bound is PMOO. However, it was demonstrated in [9] that PMOO does not necessarily outperform SFA. These problems all aggravate in the analysis of entire networks, where accurately bounding cross-traffic is important.

In order to better understand the differences between the different network analysis methods previously described, we propose in this section to numerically investigate the bounds produced by each method on a dataset of 44 044 topologies[1]. Our methodology for evaluating the methods is as follows. We generated random feed-forward networks as illustrated in Figure 1, where up to 10 servers are connected in a daisy chain manner. For each server, a rate-latency curve is used with the rate and latency sampled from a uniform distribution. Up to 40 flows are then randomly generated with random sources and destination servers, with a token bucket arrival curve with the rate and burst sampled from a uniform distribution. Each topology is then evaluated using DiscoDNC [8] (version 2.4.0) using TFA, SFA and PMOO, with different arrival bounding methods. The arrival bounding methods are labeled as: *PMOO-AB* for `ArrivalBoundMethod.PMOO` in DiscoDNC, *PMOO-PF-AB* for `PER_FLOW_PMOO`, *PBOO-PF-AB* for `PBOO_CONCATENATION`, and *PBOO-PH-AB* for `PBOO_PER_HOP`. We refer to [8] for a complete explanations of those arrival bounding methods. The topologies are built such that they satisfy the feed-forward property, i.e., there are no cyclic dependencies between the flows. For our evaluation, we focus in this paper on end-to-end delay bounds.

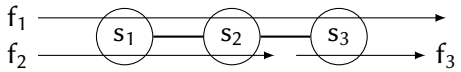

Figure 1: Example feed-forward network

The numerical results and comparison between analysis methods is presented in Figure 2. We first compare in Figure 2(a) and (b) the flows' end-to-end delay bounds against the best and worst delay bounds produced by the other methods. As expected, PMOO produces the tightest delay bounds for around 70 % of the analyzed flows, SFA for 54 %, and TFA for 1 %. Those numerical results are in line with [9, 7]. Surprisingly, we notice that PMOO produces the worst delay bounds in around 11 % of the studied flows, highlighting and confirming the existence of corner-cases. We also notice that the cross-traffic arrival bounding method has a noticeable influence on tightness.

Based on those results, we investigate in Figure 2(c) how much tightness is lost in case the network analysis does not

[1] Available here: `https://github.com/fabgeyer/dataset-itc30nc`

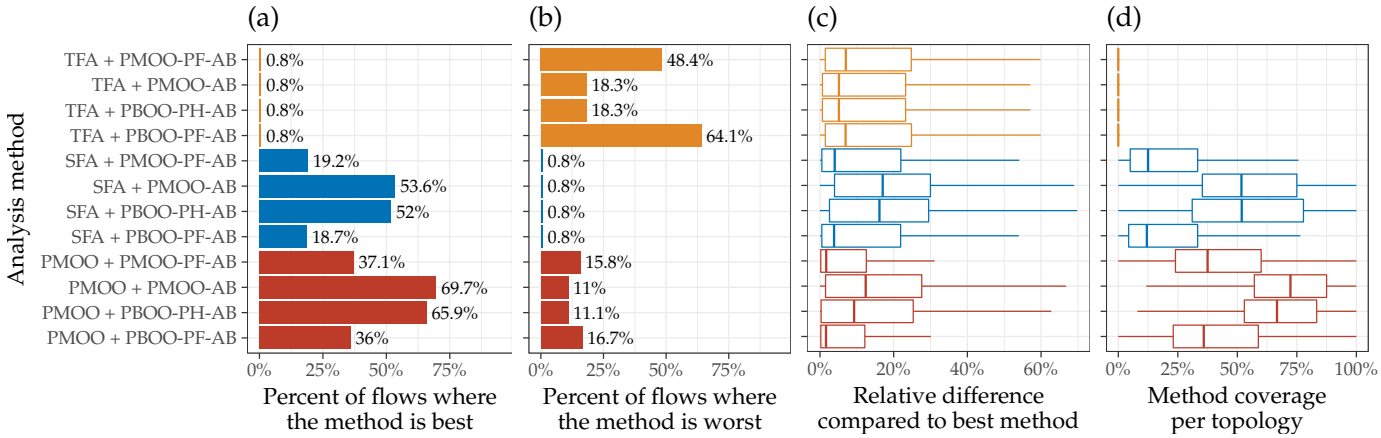

Figure 2: Evaluation and comparison of the different network analysis methods against different metrics. *(a)* Ratio of flows where a given method produces the tightest bound compared to the other methods, *(b)* respectively the worst bound. *(c)* Relative difference between delay bound of a given method and the best method when the given method does not provide the best bound. *(d)* Ratio of networks where a given method is able to produce the tightest bounds for all the flows of the topology.

produce the tightest bound. For each flow $\mathcal{F}$ where a given method M did not produce the tightest bound, we use the relative difference to numerically assess this lost tightness:

$$\textit{Relative Difference}(\mathcal{F}, M) = \frac{D_{\mathcal{F}}^{M} - \min_m D_{\mathcal{F}}^{m}}{\min_m D_{\mathcal{F}}^{m}} \quad (1)$$

with $D_{\mathcal{F}}^{M}$ the end-to-end delay bound of flow $\mathcal{F}$ with analysis method M.

Finally, we evaluate the ability of a network analysis method to produce the tightest bounds for all flows in a given network topology. To numerically assess this notion, we define the *coverage per topology* for a given method as the ratio of number of flows where the method produced the tightest bounds divided by the total number of flows in the topology. Results are presented in Figure 2(d). Although the results are in line with the numerical results from Figure 2(a) and (b), we notice that given a network topology, using a single network analysis method will not produce the tightest bounds for all flows in the topology. This finding motivates an adaptive analysis for a given topology, where different network analysis methods would be used depending on the flow of interest.

III. DNC Heuristic using Neural Networks

We introduce in this section a heuristic based on the concept of Graph Neural Network introduced in [10, 11]. The main intuition behind our approach is to map network topologies and flows to graphs. Those graph representations are then used as input for a neural network architecture able to process general graphs.

A. Presentation

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with nodes $v \in \mathcal{V}$ and edges $e \in \mathcal{E}$. Let $\mathbf{i}_v$ and $\mathbf{o}_v$ represent respectively the input features and target values of node v. The concept behind Graph Neural Networks is called *message passing*, where hidden representations of nodes are based on the hidden representations of their neighboring nodes. Those hidden representations are propagated through the graph using multiple iterations until a fixed point is found. The final hidden representation is then used for predicting a property about the node. This concept can be expressed as:

$$\mathbf{h}_v^{(t)} = f\left(\left\{\mathbf{h}_u^{(t-1)} \;\middle|\; u \in \textsc{Nbr}(v)\right\}\right) \quad (2)$$

$$\mathbf{o}_v = g\left(\mathbf{h}_v^{(t\to\infty)}\right) \quad (3)$$

$$\mathbf{h}_v^{(t=0)} = \mathit{init}\left(\mathbf{i}_v\right) \quad (4)$$

with $\mathbf{h}_v^{(t)}$ representing the hidden representation of node v at time t, $f(\cdot)$ a function which aggregates the different hidden representations, $\textsc{Nbr}(v)$ the set of neighboring nodes of v, $g(\cdot)$ a function for transforming the final hidden representation to the target values, and $\mathit{init}(\cdot)$ a function for initializing the hidden representations based on the input features.

The concrete implementation of the $f(\cdot)$ and $g(\cdot)$ functions are feed-forward neural networks with the special case that $f(\cdot)$ in Equation (2) is the sum of per-edge terms (as recommended by [11]) such that:

$$\mathbf{h}_v^{(t)} = f\left(\left\{\mathbf{h}_{\textsc{Nbr}(v)}^{(t-1)}\right\}\right) = \sum_{u \in \textsc{Nbr}(v)} f^*\left(\mathbf{h}_u^{(t-1)}\right) \quad (5)$$

with $f^*(\cdot)$ a feed-forward neural network. For $\mathit{init}(\cdot)$, the initial features are usually zero-padded to fit the dimensions of the hidden representations.

B. Extensions

We give in this section a brief overview of the extensions to GNNs which were used in this paper.

1) Gated Graph Neural Network: In order to improve the training of Graph Neural Networks, Li et al. [12] proposed Gated Graph Neural Networks (GG-NNs). This extension implements the function $f(\cdot)$ using a memory unit called Gated Recurrent Unit (GRU) [13] and unrolls Equation (2)

for a fixed number of iterations. The propagation of hidden representations among neighboring nodes for one time-step is formulated as:

$$\mathbf{x}^{(t)} = \mathbf{H}^{(t-1)}\mathbf{A} + \mathbf{b}_a \tag{6}$$

$$\mathbf{z}^{(t)} = \sigma\left(\mathbf{W}_z x^{(t)} + \mathbf{U}_z \mathbf{H}^{(t-1)} + \mathbf{b}_z\right) \tag{7}$$

$$\mathbf{r}^{(t)} = \sigma\left(\mathbf{W}_r x^{(t)} + \mathbf{U}_r \mathbf{H}^{(t-1)} + \mathbf{b}_r\right) \tag{8}$$

$$\widetilde{\mathbf{H}}^{(t)} = \tanh\left(\mathbf{W} x^{(t)} + \mathbf{U}\left(\mathbf{r}^{(t)} \odot \mathbf{H}^{(t-1)}\right) + \mathbf{b}\right) \tag{9}$$

$$\mathbf{H}^{(t)} = \left(1 - \mathbf{z}^{(t)}\right) \odot \mathbf{H}^{(t-1)} + \mathbf{z}_v^{(t)} \odot \widetilde{\mathbf{H}}^{(t)} \tag{10}$$

where $\sigma(x) = 1/(1+e^{-x})$ is the logistic sigmoid function and $\odot$ the element-wise matrix multiplication. $\{\mathbf{W}_z, \mathbf{W}_r, \mathbf{W}\}$ and $\{\mathbf{U}_z, \mathbf{U}_r, \mathbf{U}\}$ are trainable weights matrices, and $\{\mathbf{b}_a, \mathbf{b}_r, \mathbf{b}_z, \mathbf{b}\}$ are trainable biases vectors. $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is the graph adjacency matrix. Equation (6) corresponds to the aggregation of messages from adjacent nodes as in Equation (5). Equations (7) to (10) correspond respectively to the reset gate, the update gate, the candidate output, and the output vector of a standard GRU cell [13].

2) Edge attention: A recent advance in neural networks has been the concept of *attention*, which provides the ability to a neural network to focus on a subset of its inputs. For the scope of GNNs, we introduce here so-called *edge attention*, namely we wish to give the ability to each node to focus only on a subset of its neighborhood. Formally, let $a_{(v,u)}^{(t)} \in [0,1]$ be the attention between node v and u. Equation (5) is then extended:

$$\mathbf{h}_v^{(t)} = \sum_{u \in \text{NBR}(v)} a_{(v,u)}^{(t)} \cdot f^*\left(\mathbf{h}_u^{(t-1)}\right) \tag{11}$$

$$\mathbf{a}_{(v,u)}^{(t)} = f_A\left(\mathbf{h}_v^{(t-1)}, \mathbf{h}_u^{(t-1)}\right) \tag{12}$$

C. Application to deterministic network calculus

In order to apply the concepts described in Sections III-A and III-B to network calculus analysis, we model the feed-forward server graph and the flows crossing it into graphs. Each servers is represented as a node in the graph, with edges corresponding to the connections between servers. Each flow is represented as a node with edges connecting it to the path of traversed servers. Since the order of the servers which is traversed by a flow plays a large influence in network calculus, so-called *path ordering* nodes are added on the edges between the flow node and the server nodes. Figure 3 illustrates this graph encoding with the network from Figure 1.

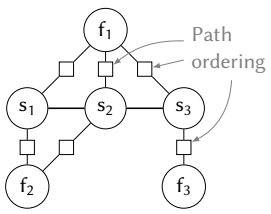

Figure 3: Graph encoding of the example topology. Square nodes represent additional nodes encoding path ordering.

Each node in the graph has the following input features:

- For server $\mathcal{S}$, we use the parameters of its rate-latency service curve: $\mathbf{i}_\mathcal{S} = [rate_\mathcal{S}, latency_\mathcal{S}]$;
- For flow $\mathcal{F}$, we use the parameters of its token bucket arrival curve: $\mathbf{i}_\mathcal{F} = [rate_\mathcal{F}, burst_\mathcal{F}]$;
- For a path ordering node $\mathcal{O}$, a categorical encoding of the hop index is used as input feature. We use standard one-hot encoding, namely $\mathbf{i}_\mathcal{O}$ is a vector with a one at the hop index, and zeros otherwise (e.g.: in Figure 3, we have $\mathbf{i}_\mathcal{O}^{f_1-s_1} = [1,0,0]$, $\mathbf{i}_\mathcal{O}^{f_1-s_2} = [0,1,0]$, $\mathbf{i}_\mathcal{O}^{f_1-s_3} = [0,0,1]$).

For the output prediction of each node representing a flow $\mathcal{F}$, we wish to have a vector of end-to-end latency bound for the 12 methods evaluated in Section II-C, namely: $\mathbf{o}_\mathcal{F} = [D_\mathcal{F}^{m_1}, D_\mathcal{F}^{m_2}, \ldots, D_\mathcal{F}^{m_{12}}]$. Similar output vectors may be used for the servers' backlog bound.

IV. Numerical evaluation

We evaluate in this section the accuracy of the proposed heuristic as introduced in Section III-C.

A. Evaluation as latency bound heuristic

We first assess in this section the precision of the predicted end-to-end latency bounds. Figure 4 illustrates the absolute relative difference between the predicted end-to-end delay bound and the bound given by the analytical method (named here *ground truth*). The overall median value is 2.5 %, with larger errors in case of predicting the output of PMOO.

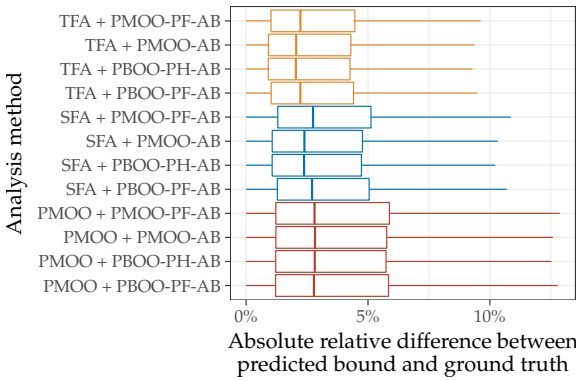

Figure 4: Precision of the predicted end-to-end latency bound

B. Evaluation as a proxy for analysis method selection

Based on the results from Section II-C, we introduce here an adaptive network analysis method, where a specific network analysis method is used given a flow of interest. This adaptive network analysis is illustrated in Algorithm 1.

Algorithm 1 Adaptive network analysis

for all flow of interest $\mathcal{F}$ in network $\mathcal{N}$ **do**
  netcalc_method $\leftarrow$ select_netcalc_method($\mathcal{N}, \mathcal{F}$)
  $bound_\mathcal{F} \leftarrow$ netcalc_method($\mathcal{N}, \mathcal{F}$)

For the function *select_netcalc_method* in Algorithm 1, we define here the following strategies:

- *Global top 1*: we always use the network analysis method which provided the best median bound in Section II-C (ie. PMOO with SFA per-hop arrival bounding);
- *Global top 2*: we evaluate the two methods which provided the best median bounds in Section II-C (ie. PMOO with SFA per-hop and PMOO global arrival bounding) and select the method which produced the tightest bound;
- *PMOO and SFA*: both PMOO and SFA with SFA per-hop arrival bound are evaluated;
- *Per-topo best*: we use the method which provided the best median bound over the flows of the studied topology;
- *Fully random*: we randomly select among the 12 methods surveyed here, with the same probability for each method;
- *Weighted random*: same as in *Fully random* but with probability values set according a ranking of the methods;
- *ML top 1*: the heuristic from Section III-C is used for selecting the analysis according to the minimum predicted end-to-end delay bound;
- *ML top 2*: as in *ML top 1*, but two analyses are selected and the one producing the minimal tightest bound is kept.

We first evaluate in Figure 5 the ability of Algorithm 1 to produce the tightest per-flow end-to-end delay bound according to the different selection strategies previously listed. The machine learning based strategies outperform all the other strategies in Figure 5, with the ability to produce the tightest result for 88 % of the studied flows for the *ML top 2* strategy, outperforming all the other evaluated strategies.

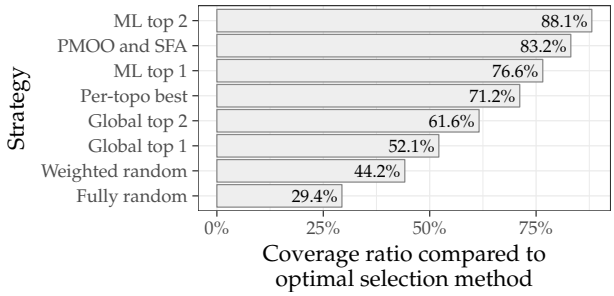

Figure 5: Ability of a selection method to produce the tightest end-to-end delay bound.

As in Figure 2(c), we evaluate in Figure 6 how much tightness is lost in case the produced end-to-end latency bound is not the tightest. Using the machine learning heuristic, we still get tight bounds compared to the other approaches listed earlier. This means that although the heuristic did not result in selecting the tightest network calculus analysis to use, the one which was selected produced a bound close to the tightest one.

Finally we evaluate in Figure 7 the gain in tightness of using an adaptive network analysis compared to only using a single analysis for all the evaluated topologies. In average, we see that we get a relative gain in tightness of around 12.79 % by using an adaptive network analysis based on machine learning, close to the optimal 13.03 %.

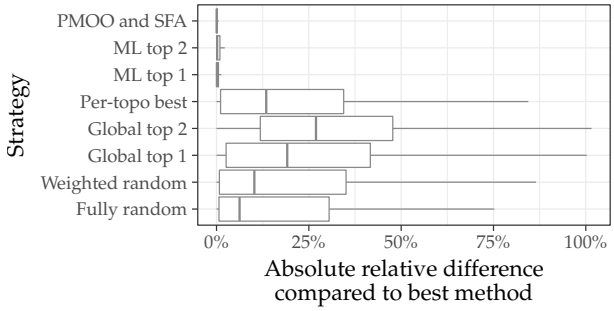

Figure 6: Relative difference between delay bound of a given method and the best method when the given method does not provide the best bound

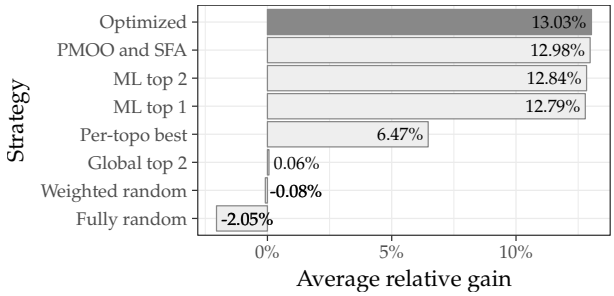

Figure 7: Relative gain in tightness compared to using only one method for all the evaluated topologies. "Optimized" corresponds here to the evaluation of the 12 network analyses.

C. Runtime

We evaluate here the runtime of using such heuristic in order to assess if the heuristic is of practical benefit regarding computation time. We evaluated the machine learning heuristic both on GPU (Nvidia GeForce GTX 1080 Ti) and CPU (Intel Xeon E3-1270). The network calculus analyses were run on the same CPU. The runtimes discussed here do not include the computation cost of training the neural network.

We compare in Figure 8 the average runtime per topology of the machine learning heuristic against the average runtimes of the different network analyses studied here. Compared to a single analysis, our heuristic is an order of magnitude faster on GPU. Compared to the sum of the runtimes of all the analyses, our heuristic is two orders of magnitude faster on GPU. This shows that our machine learning heuristic is both the best performing regarding accuracy as showed in Figures 5 and 7, but can also be used at little computational cost on GPU.

V. Related work

Identifying corner-cases of network calculus has already attracted some previous work. Schmitt et al. [9] showed that PMOO suffers from shortcomings given specific network configurations, and provided an optimization-based analysis that implements all three analysis principles at the same time. However, Kiefer et al. [14] showed that this optimization method suffers from vast computational effort. Bouillard et al. proposed in [4] another attempt to solve this challenge is using

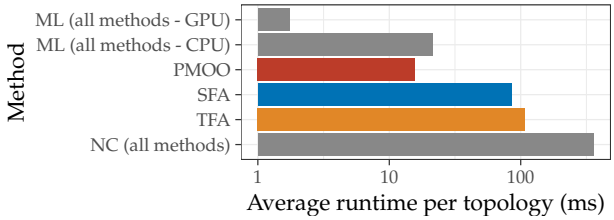

Figure 8: Average runtimes per topology of the different network analysis and arrival bounding methods, and the machine learning heuristic.

optimization-based analysis, but it was similarly showed by Bondorf et al. [15] to become computationally infeasible.

Various proposals have been made in order to make network calculus more computationally efficient. Luangsomboon et al. [16] proposed to use GPUs for computing fast convolution and deconvolution. While this approach provided efficient operations of the $(\min, +)$ algebra, the benefit in the case of network analysis is still to be determined. Bondorf et al. [15] recently proposed another approach based on exhaustive decomposition of network. Numerical evaluation showed that this method could reach bounds comparable to the ones from optimization-based methods at lower computational cost.

Neural networks for graphs has recently attracted a larger interest, and are generally based on the concept of message passing presented in Section III. They have been used in a variety of domains such as performance evaluation of networks with TCP flows [17], routing protocols [18], or basic logical reasoning tasks and program verification [12].

VI. Conclusion

Through a numerical evaluation on randomly generated networks of various network analysis methods from DNC, we showed and confirmed the existence of corner-cases, highlighting that the choice of which method to use given a network and a flow of interest is not trivial. This motivated our case for having a DNC heuristic able to predict which network analysis method will produce the tightest bound.

We contributed in this paper a novel heuristic for deterministic network calculus using graph-based deep learning. Our approach is based on the application of Graph Neural Networks and a mapping from feed-forward server graphs and the flows crossing them to graphs which can be used for training a neural network. We showed via a numerical evaluation that our approach is able to reach good accuracies and predict which network analysis will produce the tightest bounds. Finally, we evaluated the runtime of our heuristic and showed that it can be used at a small computational cost compared to traditional network analyzes.

Acknowledgments

The authors would like to thank Steffen Bondorf for his feedback on an early version of this paper. This work was supported by the German Federal Ministry of Education and Research (grant 16KIS0538, project DecADe), by the German-French Academy for the Industry of the Future, and the High-Performance Center for Secure Networked Systems.

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, Feb. 2016.

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

[3] J. Schmitt, F. A. Zdarsky, and I. Martinovic, "Improving Performance Bounds in Feed-Forward Networks by Paying Multiplexing Only Once," in *Proceedings of the 14th GI/ITG Conference on Measurement, Modeling, and Evaluation of Computer and Communication Systems (MMB 2008)*, Mar. 2008, pp. 1–15.

[4] A. Bouillard, L. Jouhet, and E. Thierry, "Tight Performance Bounds in the Worst-Case Analysis of Feed-Forward Networks," in *INFOCOM 2010*. IEEE, Mar. 2010.

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

[6] S. Bondorf, P. Nikolaus, and J. B. Schmitt, "Catching Corner Cases in Network Calculus – Flow Segregation Can Improve Accuracy," in *Proceedings of 19th International GI/ITG Conference on Measurement, Modelling and Evaluation of Computing Systems*, Feb. 2018.

[7] S. Bondorf and J. B. Schmitt, "Calculating accurate end-to-end delay bounds – you better know your cross-traffic," in *Proceedings of the 9th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2015)*, Dec. 2015.

[8] ——, "The DiscoDNC v2 – A Comprehensive Tool for Deterministic Network Calculus," in *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2014)*, Dec. 2014.

[9] J. B. Schmitt, F. A. Zdarsky, and M. Fidler, "Delay Bounds under Arbitrary Multiplexing: When Network Calculus Leaves You in the Lurch…," in *Proceedings of the 27th Annual Joint Conference of the IEEE Computer and Communications Societies*, ser. INFOCOM 2008, Apr. 2008, pp. 1669–1677.

[10] M. Gori, G. Monfardini, and F. Scarselli, "A New Model for Learning in Graph Domains," in *Proceedings of the 2005 IEEE International Joint Conference on Neural Networks*, ser. IJCNN'05, vol. 2. IEEE, Aug. 2005, pp. 729–734.

[11] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The Graph Neural Network Model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

[12] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated Graph Sequence Neural Networks," in *Proceedings of the 4th International Conference on Learning Representations*, ser. ICLR'2016, Apr. 2016.

[13] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," Jun. 2014.

[14] A. Kiefer, N. Gollan, and J. Schmitt, "Searching for tight performance bounds in feed-forward networks," *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, pp. 227–241, 2010.

[15] S. Bondorf, P. Nikolaus, and J. B. Schmitt, "Quality and cost of deterministic network calculus – design and evaluation of an accurate and fast analysis," *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, vol. 1, no. 1, p. 34, 2017.

[16] N. Luangsomboon, R. Hesse, and J. Liebeherr, "Fast Min-plus Convolution and Deconvolution on GPUs," in *Proceedings of the 11th International Conference on Performance Evaluation Methodologies and Tools*, ser. VALUETOOLS 2017, Dec. 2017.

[17] F. Geyer, "Performance Evaluation of Network Topologies using Graph-Based Deep Learning," in *Proceedings of the 11th International Conference on Performance Evaluation Methodologies and Tools*, ser. VALUETOOLS 2017, Dec. 2017, pp. 20–27.

[18] F. Geyer and G. Carle, "Learning and Generating Distributed Routing Protocols Using Graph-Based Deep Learning," in *Proceedings of the 2018 SIGCOMM Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, ser. Big-DAMA 2018. Budapest, Hungary: ACM, Aug. 2018, pp. 40–45.