DeepTMA: Predicting Effective Contention Models for Network Calculus using Graph Neural Networks

Fabien Geyer*
Technical University of Munich | Airbus CRT
Munich, Germany

Steffen Bondorf†
NTNU – Norwegian University of Science and Technology
Trondheim, Norway

***Abstract*—Network calculus computes end-to-end delay bounds for individual data flows in networks of aggregate schedulers. It searches for the best model bounding resource contention between these flows at each scheduler. Analyzing networks, this leads to complex dependency structures and finding the tightest delay bounds becomes a resource intensive task. The exhaustive search for the best combination of contention models is known as Tandem Matching Analysis (TMA). The challenge TMA overcomes is that a contention model in one location of the network can have huge impact on one in another location. These locations can, however, be many analysis steps apart from each other. TMA can derive delay bounds with high degree of tightness but needs several hours of computations to do so. We avoid the effort of exhaustive search altogether by predicting the best contention models for each location in the network. For effective predictions, our main contribution in this paper is a novel framework combining graph-based deep learning and Network Calculus (NC) models. The framework learns from NC, predicts best NC models and feeds them back to NC. Deriving a first heuristic from this framework, called DeepTMA, we achieve provably valid bounds that are very competitive with TMA. We observe a maximum relative error below 6 %, while execution times remain nearly constant and outperform TMA in moderately sized networks by several orders of magnitude.**

I. Introduction

A. Motivation

Deterministic performance bounds have seen many applications in modern systems and a wide range of network calculus-based solutions have been proposed. Network Calculus (NC) can be applied to ensure deadlines in networks for x-by-wire applications [1] as well as SDN-enabled networks [2], for safety-critical production systems [3], or both of these [4]. Moreover, NC solutions have been proposed for highly dynamic environments. E.g., admission control in self-modeling sensor networks [5] or systems providing customers with service level agreements [6] for, among others, storage access [7]. Other recent examples where dynamic events may often cause changes are cache networks [8] and cloud computing [9]. These areas benefit from fast computations of tight performance bounds. The literature provides one-shot analyses for topology-agnostic bounds [10] or bounds that hold for the specification's worst case [4]. Yet, these attempts are ultimately paid for with wasted resources. Our approach does not compromise on bound quality by providing a fast analysis that considers all details of the analyzed network.

* This work was supported by the German-French Academy for the Industry of the Future.

†This work was partially carried out during the tenure of a Carl-Zeiss Foundation fellowship in the DISCO Labs at TU Kaiserslautern, Germany, and partially during the tenure of an ERCIM 'Alain Bensoussan' Fellowship Programme at NTNU Trondheim, Norway.

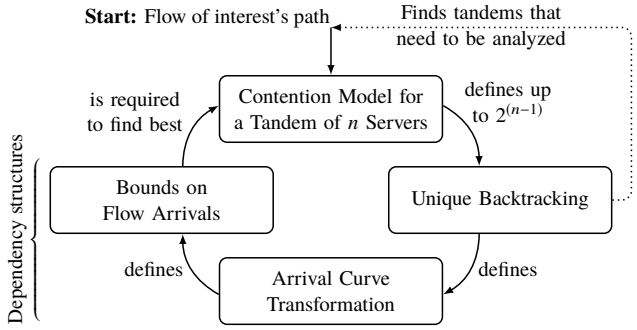


Figure 1: Dependency cycle defining current NC analyses.

B. Background

In network calculus, a network needs to be modeled by servers (e.g. queues or packet schedulers) whose forwarding capabilities are lower bounded by service curves. These curves are derived for each server's respective aggregate scheduler. Data flows traverse sequences of servers where they compete for resources with other flows. Multiplexing and reordering in queues can occur arbitrarily but deterministic bounds can be computed as data arrivals are bounded by arrival curves. The arrival curves are, however, only known at a flow's first server.

Given such a network model, the NC analysis computes a bound on an individual flow's end-to-end delay. This flow is known as flow of interest (foi) and NC must derive a model for resource contention from this flow's point of view. NC offers multiple network analysis methods to derive contention models that discriminate against the foi. These alternatives are all proven to result in valid delay bounds for the foi. But there is not a single-best contention model that can be expressed with NC, not even on a simple tandem of servers. All the worse for NC, it needs to bound the impact of all transformations of all flows' arrival curves up to the location of contention to rank the contention models. Curve transformations, in turn, require to backtrack all flows, either in an aggregate or separated by worst-case priority assumptions. Different contention models require different flow aggregation/separation assumptions and

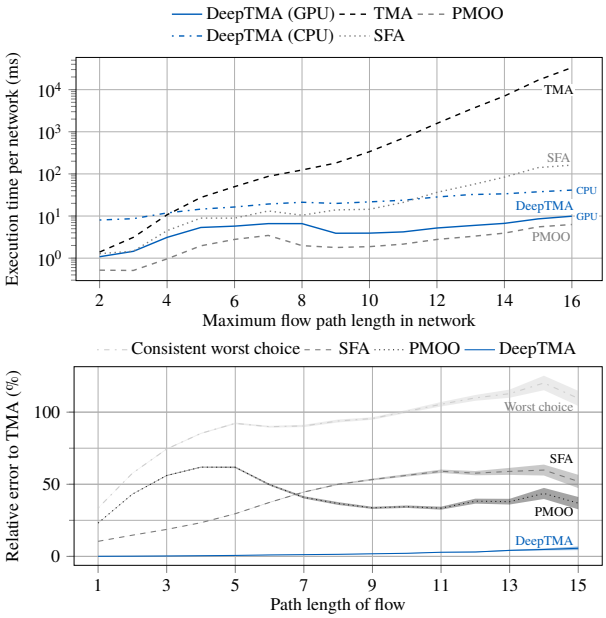


Figure 2: Comparison of DeepTMA to existing NC heuristics.

the resulting structures expressing dependencies of NC operations become unique. This cycle is shown in Figure 1. An analysis must execute at least one complete recursion, terminating upon reaching all backtracked flows' sources.

It was shown that it is possible to exhaustively derive all dependency structures and rank each contention model on each tandem occurring in a network analysis. This is known as the Tandem Matching Analysis (TMA) [11]. It achieves high degrees of delay bound tightness by enumerating all contention models upstream from the foi. Thus, the best model for a downstream location and flow can be found. In other words, TMA unwinds all loops that can be taken in the cycle in Figure 1. This is very costly. The amount of alternative contention models on a single tandem of n servers is $2^{(n-1)}$. TMA provides a recursive algorithm whose execution time can exceed several hours, e.g., when analyzing networks with >1000 servers and four times as many flows on many-core platforms that compromise on per-core performance [11].

In this paper, we present the deep-learning assisted TMA, DeepTMA, that predicts the best contention model with high efficacy, resulting in a high degree of delay bound tightness although we only start a single backtracking in Figure 1. Single backtrackings have been attempted before, yet, we are the first to achieve considerably faster execution times than TMA without considerably compromising on delay bound tightness.

C. Contributions

While we focus our evaluations on the novel DeepTMA heuristic for NC's tandem matching analysis, we contribute an entire underlying framework that combines the theories of NC [12] and a graph-based deep learning, namely Graph Neural Networks (GNNs) [13], as well as two of their respective tools [14, 15]. DeepTMA achieves the following properties:

a) Deterministic bounds: We learn from NC and feed predictions back to NC. We predict the best choices for decisions made during the NC analyses. NC stays in control and guarantees provably correct bounds.

Our deep learning framework does not learn to predict a delay bound but it predicts the most important decisions within an analysis, the contention models. Compared to directly predicting a flow's delay bound, our approach always guarantees for a valid worst-case bound as we continue to apply the proven NC operations in their valid orders.

b) Fast execution times and high tightness: Figure 2 shows first benchmarking results of DeepTMA. We compare against TMA and the established SFA [12] and PMOO [16] heuristics of NC. These are fast as they greedily decide on a single contention model, ignoring arrival and service curves. DeepTMA from our framework is minimally slower than PMOO but faster than SFA and TMA. Moreover, recent work [17] showed that the TMA cannot be parallelized easily and a speedup of only one order of magnitude was observed. In terms of delay bound tightness (relative error to TMA), all heuristics outperform a consistent worst choice of contention models. DeepTMA-derived delay bounds are tightest among these heuristics, deviating from TMA by no more than 6 % in our experiments.

DeepTMA efficacy beating SFA and PMOO in the cost/tightness-tradeoff is necessary, yet, by no means sufficient to conclude that our deep-learning assisted analysis framework is the best alternative to create heuristics. SFA and PMOO were created a decade before TMA, i.e., they never benefited from advances that resulted in TMA. Therefore, we base our statement on numerical evaluations benchmarking against newly contributed non-deep-learning TMA heuristics from the NC framework.

c) Train once, apply infinitely often: Naturally, we only train our machine learning part once before using its predictions in DeepTMA. While we chose a reasonably large range of parameters for arrival and service curves to learn from, we restricted our dataset to simple topologies (tandems and sink trees). The results shown above are achieved by predicting the best contention model for bounding each flow's delay in different, independently created tandems and sink trees.

d) Portability: While we combined two existing tools whose dependencies must be met, our combination of both theories enforces no dependencies. It is generally portable to any platform used in an area mentioned in the beginning. For instance, Figure 2 shows results for execution on CPU or GPU. Moreover, efficient deep learning libraries are becoming increasingly available in a variety of programming languages.

D. Outline

The remainder of the paper is organized as follows: Section II presents the NC theory, covering modeling and the TMA, that we will express as a graph analysis task and combine with GNNs in Section III. In Section IV we present the combination of tools and the generation of a dataset to learn from. Section V provides new NC heuristics in order

to benchmark DeepTMA against modern non-deep-learning approaches in Section VI. Section VII presents the related work on our research direction for network calculus and graph neural networks before Section VIII concludes our work and gives an outlook.

II. Network Calculus

NC models resource provision and demand with non-negative, wide-sense increasing functions from the set

$$\mathcal{F}_0 = \left\{ f : \mathbb{R} \to \mathbb{R}_\infty^+ \;\middle|\; f(0) = 0,\ \forall s \le t : f(s) \le f(t) \right\},$$

$\mathbb{R}_\infty^+ := [0, +\infty) \cup \{+\infty\}$. Functions of $\mathcal{F}_0$ are also used for the bounding curves of NC. Arrival curves upper bound data arrivals and service curves lower bound forwarding guarantees.

Definition 1 (Arrival Curve): Let the data arrivals of a flow over time be characterized by function $A(t) \in \mathcal{F}_0$, where $t \in \mathbb{R}_\infty^+$. Then, an arrival curve $\alpha(d) \in \mathcal{F}_0$ for $A(t)$ must fulfill

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

i.e., it must bound the flow's data arrivals in any duration d.

Definition 2 ((Strict) Service Curve): If, during any period with backlogged data of duration d, a server s with input function A guarantees an output of at least $\beta(d) \in \mathcal{F}_0$, then it is said to offer a (strict) service curve β.

The network calculus was cast in a (min,+)-algebra [12] with the following operations:

Definition 3 ((min,+) Operations): Network calculus applies $(\min, +)$-algebraic operations to compute curve transformations bounding the worst-case outcome of certain scenarios:
- *Flow Aggregation:* $(\alpha_1 + \alpha_2)(d) = \alpha_1(d) + \alpha_2(d)$
- *Server Crossing:* $(\alpha \oslash \beta)(d) = \sup_{u \ge 0} \{\alpha(d+u) - \beta(u)\}$
- *Residual Service:* $(\beta \ominus \alpha)(d) = \sup_{0 \le u \le d} \{\beta(u) - \alpha(u)\}$
- *Server Concat.:* $(\beta_1 \otimes \beta_2)(d) = \inf_{0 \le u \le d} \{\beta_1(d-u) + \beta_2(u)\}$

where $\alpha, \alpha_1, \alpha_2$ are arrival and β, β_1, β_2 are service curves.

These curve transformations guarantee for deterministic results. For a network such as the tandem shown in Figure 3, there are multiple valid orders of operations. Each corresponds to a model of contention that imposes a dependency structure. That structure, in turn, defines contention models upstream.

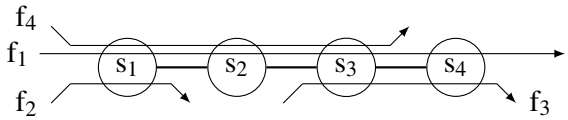


Figure 3: Example tandem network in the NC model.

Definition 4 (Contention Model): The network calculus contention model for a tandem of servers defines its orders of operations that provide a residual service guaranteed to a flow crossing said tandem. Any concatenation of sub-tandem residual service is a valid contention model for the tandem.

Suppose f_3 in Figure 3 is the flow of interest to be analyzed. Its *tandem decompositions* are defined by subtandem-separating *cuts* located between crossed servers:

all cuts: decomposition into 1-server tandems s_3 and s_4 or
no cuts: entire 2-server tandem consisting of s_3 and s_4.

On any tandem, any number and placement of cuts results in a valid tandem decomposition. Their exhaustive enumeration is known as Tandem Matching Analysis (TMA) [11]. The two cases shown here are special. The former corresponds to the classical Separated Flow Analysis (SFA) [12]. It concatenates per-server residual service bounds by computing $(\beta_3 \ominus (\alpha_{3,1} + \alpha_{3,4})) \otimes (\beta_4 \ominus \alpha_{4,1})$, where the first index denotes server location and the second one of arrival curves is the flow id. The latter contention model is known as the Pay Multiplexing Only Once (PMOO) [16] analysis that computes f_3's residual service for the concatenated tandem: $(\beta_3 \otimes \beta_4) \ominus' \{\alpha_{3,1}, \alpha_{3,4}\}$. Note the adapted residual service operation from [11] and the set of separated flows it subtracts.

Next, we need to bound the arrivals of flows to an analyzed tandem as required for the residual service curve computation under a specific contention model.

Definition 5 (Dependency Structure): A dependency structure is a set of sequences that bound arrival curve transformations up to a tandem of servers.

The dependency structure is subject to the contention model's requirements regarding flow aggregation, separation and duplication. It is created by unwinding the cycle shown in Figure 1; potentially under contention modeling restrictions (SFA or PMOO). In our example, SFA can bound f_1 and f_4 aggregately up to s_3, i.e., on the tandem of servers s_1 and s_2. Note, that either of the two decompositions as above can be best due to the interference of f_2. Additionally, SFA needs a separate arrival curve for f_1 at s_4 – it is not possible to separate flows after their aggregate crossing of a server was bounded and we do not assume additional network elements alleviating this problem like per-flow shapers [12] or interleaved traffic regulators [18]. PMOO depends on separate bounds for f_1 and f_4 for the same reason. Both flows cross the s_1,s_2-tandem and can thus benefit from either contention model due to f_2. Yet, PMOO does not check the SFA contention model / tandem decomposition. Last, note that TMA computes results for all these contention models and dependency structures to find the best one. Depending on the employed hardware, executing the TMA can take multiple hours. For example, analyzing a network with about 1500 servers and four times as many flows was shown to take close to 2 hours on a compute server [11]. Therefore, we aim to avoid this huge effort with predictions.

III. Graph Neural Network for NC

We develop our DeepTMA heuristic in this section. It is based on the concept of Graph Neural Network (GNN) introduced in [13, 19]. The goal of DeepTMA is to predict the best tandem decompositions, i.e., contention models, to use in TMA. We define networks to be in the NC modeling domain and to consist of servers, crossed by flows. We refer to the model used in GNN as graphs. The main intuition is to transform the networks into graphs. Those graph representations are then used as inputs for a neural network architecture

able to process general graphs, which will then predict the tandem decomposition resulting in the best residual service curve. Our approach is illustrated in Figure 4. Since the delay bounds are still computed using the formal network calculus analysis, they inherit their provable correctness.

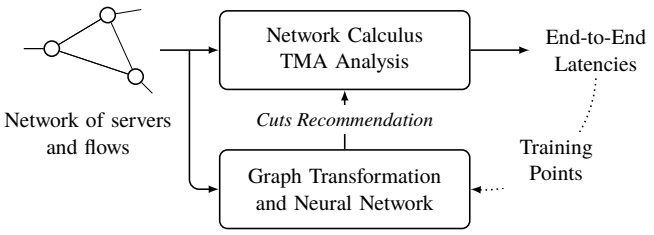


Figure 4: Overview of the proposed approach.

A. Overview of Graph Neural Networks

In this section, we detail the neural network architecture used for training neural networks on graphs, namely the family of architectures based on GNNs [13, 19].

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with nodes $v \in \mathcal{V}$ and edges $(v, u) \in \mathcal{E}$. Let $\mathbf{i}_v$ and $\mathbf{o}_v$ represent respectively the input features and output values for node v. The concept behind GNNs is called *message passing*, where hidden representations of nodes $\mathbf{h}_v$ are iteratively passed between neighboring nodes. Those hidden representations are propagated throughout the graph using multiple iterations until a fixed point is found. The final hidden representation is then used for predicting properties about nodes. This concept can be formalized as:

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

$$\mathbf{o}_v = out\left(\mathbf{h}_v^{(t\to\infty)}\right) \tag{2}$$

$$\mathbf{h}_v^{(t=0)} = init\left(\mathbf{i}_v\right) \tag{3}$$

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

The concrete implementations of the $aggr$ and out functions are feed-forward neural networks (FFNN), with the addition that $aggr$ is the sum of per-edge terms [19], such that:

$$\mathbf{h}_v^{(t)} = aggr\left(\left\{\mathbf{h}_{\textsc{Nbr}(v)}^{(t-1)}\right\}\right) = \sum_{u\in\textsc{Nbr}(v)} f\left(\mathbf{h}_u^{(t-1)}\right) \tag{4}$$

with f a FFNN. For $init$, a one-layer FFNN is used to fit the input features to the dimensions of the hidden representations.

Gated Graph Neural Networks (GGNN) [20] were recently proposed as an extension of GNNs to improve their training. This extension implements f using a memory unit called Gated Recurrent Unit (GRU) [21] and unrolls Equation (1) for a fixed number of iterations. This simple transformation allows for commonly found architectures and training algorithms for standard FFNNs as applied in computer vision or natural language processing. The neural network architecture is illustrated in Figure 5.

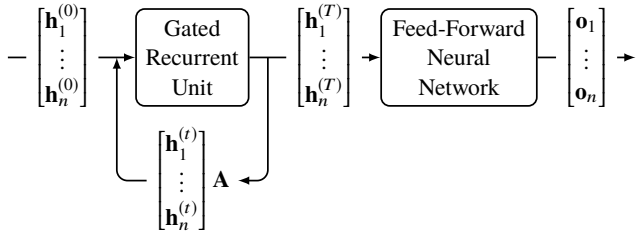


Figure 5: Gated-graph neural network architecture.

Formally, the propagation of the hidden representations among neighboring nodes for one time-step is formulated as:

$$\mathbf{x}^{(t)} = \mathbf{H}^{(t-1)}\mathbf{A} + \mathbf{b}_a \tag{5}$$

$$\mathbf{z}^{(t)} = \sigma\left(\mathbf{W}_z x^{(t)} + \mathbf{U}_z\mathbf{H}^{(t-1)} + \mathbf{b}_z\right) \tag{6}$$

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

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

$$\mathbf{H}^{(t)} = \left(1 - \mathbf{z}^{(t)}\right) \odot \mathbf{H}^{(t-1)} + \mathbf{z}_v^{(t)} \odot \widetilde{\mathbf{H}}^{(t)} \tag{9}$$

where $\sigma(x) = 1/(1+e^{-x})$ is the logistic sigmoid function and $\odot$ is the element-wise matrix multiplication. $\mathbf{W}_z, \mathbf{W}_r, \mathbf{W}$ and $\mathbf{U}_z, \mathbf{U}_r, \mathbf{U}$ are trainable weight matrices, and $\mathbf{b}_a, \mathbf{b}_r, \mathbf{b}_z, \mathbf{b}$ are trainable bias vectors. $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is the graph adjacency matrix, determining the edges in the graph $\mathcal{G}$.

Equation (5) corresponds to one time-step of the propagation of the hidden representation of neighboring nodes to node v, as formulated previously for GNNs in Equations (1) and (4). Equations (6) to (9) correspond to the mathematical formulation of a GRU cell [21], with Equation (6) representing the GRU reset gate vector, Equation (7) the GRU update gate vector, and Equation (9) the GRU output.

In order to propagate the hidden representations throughout the complete graph, a fixed number of iterations of Equations (6) to (9) are performed. This extension has been shown to outperform standard GNN which require to run the iteration until a fixed point is found.

We also extended our neural network architecture with an attention mechanism similar to the one proposed in [22]. Thus, the neural network can give preference to some neighbors over other ones via a trained function. For each edge (v, u) in the graph, we define a weight parameter $\rho_{v,u}^{(t)}$ depending on the concatenation of $\mathbf{h}_v^{(t)}$ and $\mathbf{h}_u^{(t)}$:

$$\rho_{v,u}^{(t)} = \sigma\left(\mathbf{W}_a\left\{\mathbf{h}_v^{(t)}, \mathbf{h}_u^{(t)}\right\} + \mathbf{b}_a\right) \tag{10}$$

with trainable weights $\mathbf{W}_a$ and bias parameters $\mathbf{b}_a$. Equation (4) can then be rewritten as

$$\mathbf{h}_v^{(t)} = \sum_{u\in\textsc{Nbr}(v)} \rho_{v,u}^{(t-1)}\, f\left(\mathbf{h}_u^{(t-1)}\right). \tag{11}$$

B. Application to TMA

In order to apply the concepts described in Section III-A to a network calculus analysis, we model NC's network into a graph. Figure 6 illustrates this graph encoding on the network from Figure 3.

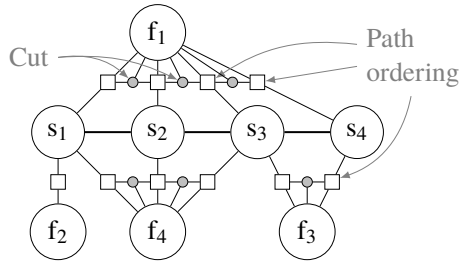


Figure 6: Transformed network of Figure 3 to the graph model.

Each server is represented as a node in the graph, with edges corresponding to the network's links. Each flow is represented as a node in the graph, too. In order to encode the path taken by a flow in this graph, we use edges to connect the flow to the servers it traverses. Since those edges do not encode the order in which those servers are traversed, so-called *path ordering* nodes are added to edges between the flow node and the traversed server nodes. This property is especially important in the TMA since the order, and hence position of cuts, has a large impact on dependency structures. In order to represent these TMA cuts, each potential cut between pairs of servers on the path traversed by the flow is represented as a node. This cut node is connected via edges to the flow and to the pair of servers it is associated to.

In addition to a categorical encoding of the node type (i.e., server, flow, path ordering or cut), the input features of each node in the graph is as follows:

- For each server s, parameters of its service curve $\beta_s(d) = \max\{0, rate_s \cdot d - latency_s\}$ are used: $[rate_s, latency_s]$
- For each flow f, parameters of its arrival curve $\alpha_f(d) = \{rate_f \cdot d + burst_f\}_{\{d>0\}}$ (i.e., $\alpha_f(d) = 0$ for $d \leq 0$) are used: $[rate_f, burst_f]$
- For each path ordering p, the hop count is encoded as a categorical one-hot vector: $[hop = 1, \ldots, hop = n]$
- Finally, cut nodes do not have input features

Note that in case more complex arrival or service curve shapes than affine curves [23] are studied, those input features can be extended to represent the additional curve parameters. Last note that edges have no features in this graph encoding.

Since the goal of our machine learning approach is to predict which tandem decomposition will result in the tightest bound, only the nodes presenting cuts have output features. We encode this problem as a classification problem, namely each cut node has to be classified in two classes: perform a cut between the pair of servers it is connected to or not: $[cut, \overline{cut}]$. The overall prediction to be fed back, i.e., the selection of one out of TMA's potential decompositions for a given foi's path, is defined by the set of all cut classifications for this path.

IV. Implementation and Dataset Generation

A. Technical Implementation

We implemented DeepTMA using Tensorflow. For the purpose of computational efficiency, passing of hidden representation between neighboring nodes was implemented with sparse operations using the graph's adjacency list instead of the graph's adjacency matrix requiring dense operations. The recursion from Equation (1) was unrolled for a fixed number of iteration according to the diameters of the analyzed graphs. Table I illustrates the size of the different layers used here.

Layer	NN Type	Size
init	FFNN	$(21, 160)_w + (160)_b$
Memory unit	GRU cell	$(320, 320)_w + (320, 160)_w + (480)_b$
Edge attention	FFNN	$(320, 1)_w + (2)_b$
out hidden layers	FFNN	$2 \times \{(160, 160)_w + (160)_b\}$
out final	FFNN	$(160, 2)_w + (2)_b$
	Total:	209 766 parameters

Table I: Size of the different layers used in the GGNN. Indexes represent respectively the weights (w) and biases (b) matrices.

We analyzed each network with the DiscoDNC [14] version 2.4. A tandem decomposition is always executed for a flow of interest. But instead of the residual service curves, we use the delay bounds for the foi as caused by all decompositions in order to rank them. This is because the former potentially faces problems in the case of lost service curve strictness.

B. Dataset Generation

In order to train our neural network architecture, we randomly generated a set of topologies, tandems like in Figure 3 and tree topologies. For each created server, a rate latency service curve was generated with uniformly random rate and latency parameters. A random number of flows with random source and sink servers was added. Note that in our topologies, there cannot be cyclic dependency between the flows. For each flow, a token bucket arrival curve was generated with uniformly random burst and rate parameters. All curve parameters were normalized to the $(0, 1]$ interval. In total, 100 000 different networks were generated, with a total of more than 2 million flows, and close to 60 million tandem decompositions. Half of the networks were generated following tandem topologies, and half following tree topologies. Table II summarizes different statistics about the generated dataset. The dataset is available online[1] to reproduce our learning results.

Parameter	Min	Max	Mean	Median
# of servers	2	41	14.2	12.0
# of flows	1	63	23.0	18.0
# of flows per server	1	44	5.8	4.6
# of tandem combinations	2	113 100	596.2	134.0
# of tandem combination per flow	2	32 768	25.9	4.0
# of nodes in analyzed graph	6	717	159.0	127.0

Table II: Statistics about the generated dataset.

[1] https://github.com/fabgeyer/dataset-infocom2019

V. TMA Heuristics from the NC Framework

As we will detail in Section VII, there is no other combination of NC and deep learning for deterministic performance analysis. Nor is there any other combination of NC or deep learning with a third methodology for fast delay bounding. To benchmark DeepTMA, we present three new heuristics for the choice of TMA's tandem decompositions. All are derived from the NC framework to showcase its potential to find the tightest end-to-end delay bound without exhaustive analysis.

A. RND: Random Choice of Tandem Decomposition

The simplest heuristic is to select multiple alternative tandem decompositions randomly following a uniform distribution. Given any n-server tandem, starting with the foi's path as shown in Figure 1, RND only selects $n' \ll 2^{(n-1)}$ decompositions. I.e., the RND heuristic randomly samples a small part of TMA's search space per tandem in the analysis. The remainder of the analysis follows the standard NC proceeding.

B. PLH: Path Length of Flows up to Location of Interference

Due to the exponential growth of the number of tandem decompositions, the chance of randomly selecting the tandem decomposition resulting in the tightest bound decreases exponentially with the number of servers traversed by the foi. In this second heuristic, we use the intuition that the probability of a cut depends on the cross-flow at each server and the fact that the arrival curve of cross-flows depends on the path traversed. In order to define it, we use h, the number of servers that each cross-flow crossed before reaching this location, and empirically fit the probability $\Pr(cut|h_{\text{avg}})$ with h_{avg} the average of h over all cross-flows. In homogeneous sink-tree networks, this heuristic can even obtain a precise ranking of flow arrival curves as well as the relative differences between them due to the lack of flow demultiplexing [5].

C. HCH: Hop Count Heuristic

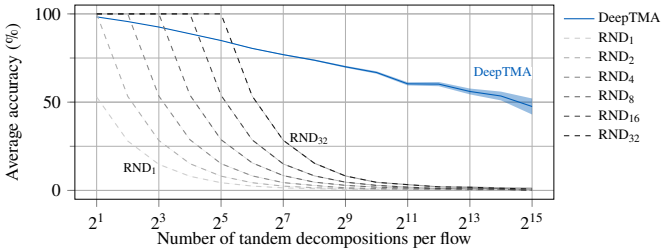
This last heuristic is based on the probability of cutting a tandem according the number of traversed servers as observed in our data set. In order to correctly parametrize this location, we empirically fit a distribution $\Pr(cut|l,p)$ predicting the best cut for each path length l and cut position p. The procedure for generating a tandem cut is illustrated in Algorithm 1. As in Section V-A, multiple tandem decompositions may be generated and evaluated.

Algorithm 1 Decomposition using experimental probability.

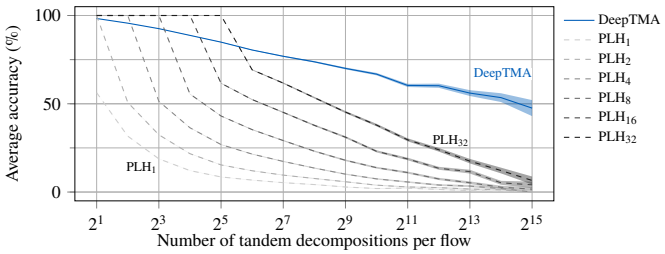
$v \leftarrow [c_1, \ldots, c_l] \sim \mathcal{U}(0,1)^l$
cuts $\leftarrow \mathbb{I}\left(v \leq [\Pr(cut|l,1), \ldots, \Pr(cut|l,l)]\right)$
($\mathbb{I}$ is the indicator function)

VI. Numerical Evaluation

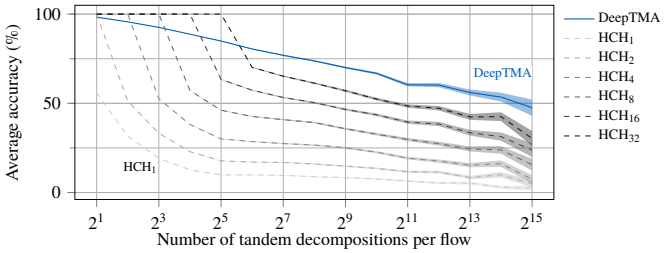
We evaluate in this section DeepTMA against the heuristics presented above. Via a numerical evaluation, we illustrate the tightness and execution time of DeepTMA and highlight its usability for practical use-cases.



(a) Comparison against RND heuristic.



(b) Comparison against PLH heuristic.



(c) Comparison against HCH heuristic.

Figure 7: Accuracy of DeepTMA and the new heuristics.

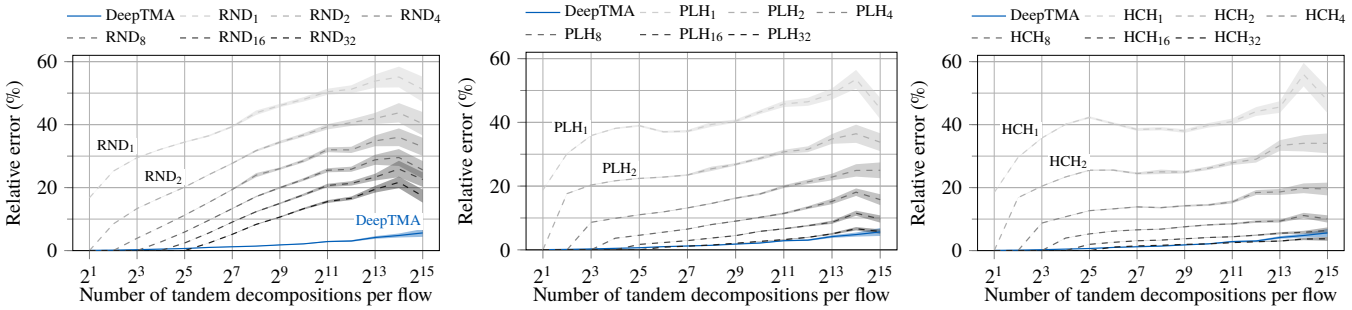
A. Prediction Accuracy

Since multiple tandem decompositions may be valid and since we know the tightest bounds from TMA, we define the accuracy for a given foi and a given method as 1 if the tandem decomposition predicted by the method resulted in the tightest delay bound, and 0 otherwise. We evaluate in Figure 7 the outcome of the different heuristics evaluated on our dataset.

Figure 7a compares DeepTMA against the RND heuristic presented in Section V-A, namely random choices of tandem decomposition. We also evaluate the case where multiple random tries are evaluated and the one leading to the tightest delay bound is kept (labeled by the index in the figure). We note in Figure 7a that DeepTMA achieves average accuracies larger than 50 % even for flows where the possible number of tandem decomposition goes up to 32 768. Compared to this heuristic, DeepTMA achieves much better accuracies for flows with a larger number of hop.

Figure 7b compares DeepTMA against the PLH heuristic presented in Section V-B, namely the cross-flow statistics heuristic. This heuristic achieves better accuracy compared to the previous one, but it still fails to reach good accuracy for networks with a large number of cuts.

Finally, Figure 7c compares DeepTMA against the HCH



(a) Comparison against RND heuristic. (b) Comparison against PLH heuristic. (c) Comparison against HCH heuristic.

Figure 8: Relative error of DeepTMA and the heuristics presented in Section V.

heuristic presented in Section V-C, namely the hop count statistics heuristic. We notice that this heuristic achieves better accuracy for networks with a larger number of hops compared to the two previous heuristics. Nevertheless, DeepTMA still achieves better results for larger topologies.

Overall, that DeepTMA achieves better prediction accuracy than the pure NC heuristics. This indicates that the choice of tandem decomposition is more complex than captured by indicators such as hop count or cross-flow statistics.

B. Relative Error

While we highlighted in the previous section that the accuracy of our machine learning approach diminishes as the number of hops becomes larger, we investigate in this section the resulting loss of tightness in case a non-optimal tandem decomposition was selected. In order to quantitatively evaluate this loss of tightness, we use the relative error, defined as

$$relative\ error_{\text{foi}} = \frac{delay_{\text{foi}}^{\text{heuristic}} - delay_{\text{foi}}^{\text{TMA}}}{delay_{\text{foi}}^{\text{TMA}}} \tag{12}$$

Since TMA always produces the tightest delay bound among the evaluated heuristics, this relative error is always positive.

Figure 8 compares DeepTMA against the other heuristics. Although the accuracy of DeepTMA dropped to 50 % for larger networks, the impact of these failures to predict the optimal decomposition only results in a relative error below 6 %. The results and comparison between DeepTMA and the other heuristics are in line with those presented in Figure 7. Only PLH_{32} and HCH_{32} are able to achieve a relative error similarly small as DeepTMA, yet, at a much larger computational cost since 32 different tandem combinations and their entire dependency structures have to be evaluated every time.

C. Execution Times

In order to understand the practical applicability of our heuristic, we evaluate in this section its execution time in different settings. We define and measure the execution time per network as the total time taken to process N networks and all its flows divided by N, without including the startup time or the time taken for initializing the network data structures.

Since DeepTMA can be executed on either CPU or GPU, we first compare both platforms and their affinity at parallelization in Figure 9. A Nvidia GTX 1080 Ti was used for the measurements on GPU, and an Intel Xeon E3-1270 v6 (at 3.80 GHz) for the ones on CPU. We first notice that the execution time grows close to linearly with the size of the network, both on CPU and GPU, which is explained by the iterations of message passing illustrated in Equation (5) according to the diameter of the studied graph. Execution on GPU results in faster computation compared to CPU for networks larger than two hops, mainly due to the better ability of GPUs of parallelizing the numerical operations used in neural networks.

Since both platforms offer multiple cores for parallel execution of multiple processes, we investigate the effect of batching, namely analyzing multiple networks in parallel. Parallelization of the mathematical operations described in Section III is automatically performed by Tensorflow. We present in Figure 9 the execution time without any batching – namely only one network is processed at once – and with batching, where the heuristic processes 64 networks at once. On both platforms, batching results in a reduction of processing time, which is relevant in use-cases where multiple network configuration have to be processed.

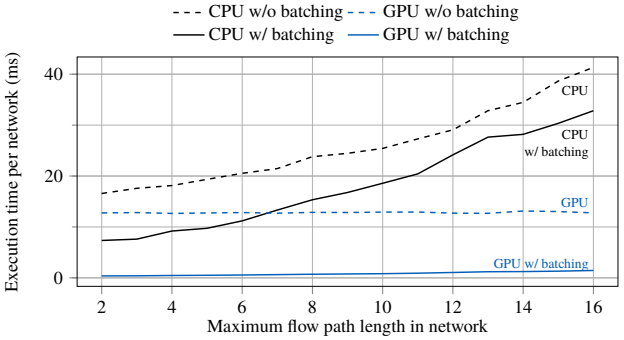


Figure 9: Execution time of the cut recommendation part of DeepTMA, executed on CPU or GPU, without batching or batch sizes of 64 networks.

In addition, we measured the execution time of TMA using

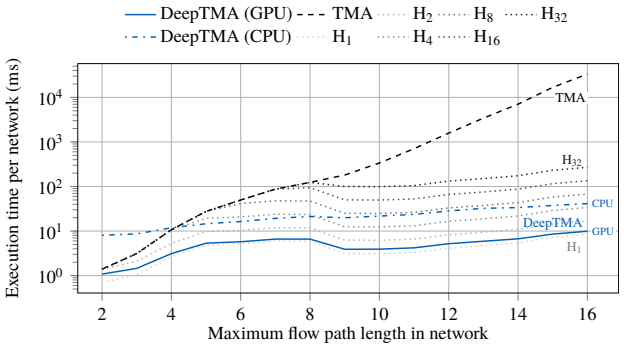


Figure 10: Execution times per topology for TMA, DeepTMA and heuristics H using n tandem decompositions (H_n).

DiscoDNC. The same CPU was used for running DiscoDNC, with Oracle's HotSpot JVM version 1.8.

Whereas Figure 9 provides insight on the computational cost of DeepTMA, Figure 10 compares it to a generalized version of the heuristics presented in Section V. Since the selection of tandem decompositions is a fast operation in all three pure NC heuristics, in particular compared to the other required operations, we only illustrate the execution time of a generic heuristic H_n selecting n decompositions per tandem. As all analyses ultimately use the DiscoDNC for the derivation of bounds, comparing the average execution times of H_n and DeepTMA (with batching), we can also judge the increase of computational effort due to our deep learning-based predictions. As expected, TMA execution times grow exponentially and H_n heuristics' execution times coincide with TMA as long as their n-value causes an exhaustive search, too. An entirely CPU-bound DeepTMA analysis is slowest in very small networks where the exhaustive enumeration of TMA is easily possible to execute. Starting at a maximum flow path length of 4, it mostly performs between H_4 and H_8. Yet, we saw in Sections VI-A and VI-B that RND_i, PLH_i and HCH_i, $i \in 4, 8$ are outperformed by DeepTMA. DeepTMA leveraging GPU technology for predictions only adds very small execution times to H_1 while achieving vastly better bounds. Compared to TMA, we can observe a measured differences in execution time growing up to four orders of magnitude.

Last, note the comparison between DeepTMA, TMA, SFA and PMOO that was already presented in Figure 2 as it highlights efficiency compared to established NC analyses.

VII. Related Work

A recent survey [24] about existing applications of machine learning to formal verification shows that this combination can accelerate formal methods, e.g., theorem proving, model-checking or SAT-SMT problems. As we show, NC has been combined with other methods, too. So have GNNs with formal verification. Yet, we are the first to combine both TMA and GNN into a framework for deterministic performance analysis.

1) Network Calculus Combined with Other Methodologies: The (min,+)-algebraic NC provides deterministic modeling and analysis techniques. It has seen various efforts to extend NC's capabilities. For instance, the underlying (min,+) algebra can be exchanged for (min,×) for fading channel analysis [25] or for (max,+) to better fit discrete event systems [26]. Moreover, a common model for NC and event stream theory has been developed [27] and state-based system modeling can be integrated by pairing NC with timed automata [28].

NC has been used to describe component models commonly found in real-time systems [29]. Delay bounds can then be derived from a combination of component characteristics and the network calculus model. For example, knowledge about the busy period of a greedy processing component has been used to speed up NC computations [30].

An optimization formulation has been derived from the NC model that computes tight bounds in networks without assumptions on the multiplexing of flows [31]. It first derives the dependencies between busy periods of servers in order to partially order the mutual impact of flows. The tight analysis requires to expand this order to all compatible total orders. This is, however, computationally infeasible. A heuristic was proposed. It uses the initially derived partial order but it, too, was shown to become computationally infeasible [11].

Recent work uses machine learning to estimate service curves from measurements [32]. In contrast to our work, this interfacing via service curves cannot compute provably correct bounds on the worst-case flow delays due to uncontrollable uncertainties introduced by measurements and machine learning.

2) Deep Learning for Graphs and Formal Verification: GNNs were first introduced in [13, 19], a concept subsequently refined in recent works. GGNNs [20] extended this architecture with modern practices by using GRU memory units [21]. Message-passing neural network were introduced in [33], with the goal of unifying various GNN and graph convolutional concepts. [22] formalized graph attention networks, which enables to learn edge weights of a node neighborhood.

These concepts were applied to many domains where problems can be modeled as graphs: chemistry with molecule analysis [34, 33], jet physics and elementary particles [35], prediction of satisfiability of SAT problems [36], or basic logical reasoning tasks and program verification [20]. For computer networks, they have recently been applied to prediction of delay bounds [37] and performance evaluation of networks with TCP flows for predicting average flow bandwidth [15, 38].

VIII. Conclusion

We contribute a new framework that combines network calculus and deep learning. The first heuristic created with our framework is the DeepTMA, deep learning-assisted TMA, a fast network analysis for deterministic end-to-end delay bounds. It solves the main bottleneck of the existing TMA, namely its exponential execution time growth with network size, by using predictions for effectively selecting the contention models in the network calculus analysis. Our work is based on a transformation of the network of servers and flows crossing them into a graph which is analyzed using

Graph Neural Networks. Via a numerical evaluation, we show that our heuristic is accurate and produces end-to-end bounds which are almost as tight as TMA. DeepTMA is as fast as or faster than previously widespread methods – namely SFA and PMOO – even when analyzing larger networks, but with a gain in tightness exceeding 50 % in some cases.

Future Work Directions: Our deep-learning assisted NC framework of Section III is already able to create other heuristics than DeepTMA. For instance, DeepTMA$_n$ with n decompositions suggested per tandem or a heuristic predicting the most computationally efficient decomposition if multiple ones will provide best results. Moreover, it can be further optimized by finding the best number of variables for the graph neural network as well as its bottleneck. Learning from a dataset including feed-forward networks, more complex curve shapes than the simple affine ones [23], or from/for FIFO-multiplexing networks [39] or entirely non-FIFO [40] is already possible, too.

Secondly, our framework is highly extensible. With some minor additions, it can predict if an optional feature of the NC analysis might be beneficial at a certain point. Examples are the alternative output bound formulation of [41] and flow prolongation [42]. Exhaustive flow prolongation is only feasible on single tandems, yet, learning from those can be sufficient as we show in our paper. Another direction is to predict a bound on the necessary domain of curves [30, 43].

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] S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, R. Yahyapour, and D. Simeonidou, "An analytical model for software defined networking: A network calculus-based approach," in *Proc. of IEEE Globecom*, 2013.

[3] X. Jin, N. Guan, J. Wang, and P. Zeng, "Analyzing multimode wireless sensor networks using the network calculus," *Journal of Sensors*, vol. 2015, Article ID 851608.

[4] J. W. Guck, A. Van Bemten, and W. Kellerer, "DetServ: Network models for real-time QoS provisioning in SDN-based industrial environments," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 4, pp. 1003–1017, 2017.

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

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

[7] T. Zhu, A. Tumanov, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, "PriorityMeister: Tail latency QoS for shared networked storage," in *Proc. of ACM SoCC*, 2014.

[8] E. J. Rosensweig and J. Kurose, "A network calculus for cache networks," in *Proc. of IEEE INFOCOM*, 2013.

[9] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: Predictable message latency in the cloud," in *Proc. of ACM SIGCOMM*, 2015.

[10] A. Charny and J.-Y. Le Boudec, "Delay bounds in a network with aggregate scheduling," in *Proc. of QoFIS*, 2000.

[11] 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.

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

[13] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proc. of IEEE IJCNN*, 2005.

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

[15] F. Geyer, "Performance evaluation of network topologies using graph-based deep learning," in *Proc. of EAI ValueTools*, 2017.

[16] J. B. Schmitt, F. A. Zdarsky, and I. Martinovic, "Improving performance bounds in feed-forward networks by paying multiplexing only once," in *Proc. of GI/ITG MMB*, 2008.

[17] A. Scheffler, M. Fögen, and S. Bondorf, "The deterministic network calculus analysis: Reliability insights and performance improvements," in *Proc. of IEEE CAMAD*, 2018.

[18] J.-Y. Le Boudec, "A theory of traffic regulators for deterministic networks with application to interleaved regulators," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2721–2733, 2018.

[19] 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, 2009.

[20] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *Proc. of ICLR*, 2016.

[21] 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," in *Proc. of EMNLP*, 2014.

[22] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. of ICLR*, 2018.

[23] A. Bouillard and É. Thierry, "An algorithmic toolbox for network calculus," *Discrete Event Dynamic Systems*, vol. 18, no. 1, 2008.

[24] M. Amrani, L. Lúcio, and A. Bibal, "ML + FV = ♡? A survey on the application of machine learning to formal verification," 2018, arxiv:1806.03600.

[25] H. Al-Zubaidy, J. Liebeherr, and A. Burchard, "A (min, ×) network calculus for multi-hop fading channels," in *Proc. of IEEE INFOCOM*, 2013.

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

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

[28] 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.

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

[30] N. Guan and W. Yi, "Finitary real-time calculus: Efficient performance analysis of distributed embedded systems," in *Proc. of IEEE RTSS*, 2013.

[31] 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.

[32] S. K. Khangura, M. Fidler, and B. Rosenhahn, "Neural networks for measurement-based bandwidth estimation," in *Proc. of IFIP Networking*, 2018.

[33] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. of NIPS*, 2017.

[34] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. of NIPS*, 2015.

[35] I. Henrion, K. Cranmer, J. Bruna, K. Cho, J. Brehmer, G. Louppe, and G. Rochette, "Neural message passing for jet physics," in *Proc. of the Deep Learning for Physical Sciences Workshop*, 2017.

[36] D. Selsam, M. Lamm, B. Bunz, P. Liang, L. de Moura, and D. L. Dill, "Learning a SAT solver from single-bit supervision," 2018, arxiv:1802.03685.

[37] F. Geyer and G. Carle, "The case for a network calculus heuristic: Using insights from data for tighter bounds," in *Proc. of NetCal*, 2018.

[38] F. Geyer, "DeepComNet: Performance evaluation of network topologies using graph-based deep learning," *Performance Evaluation*, 2018.

[39] A. Bouillard and G. Stea, "Exact worst-case delay in FIFO-multiplexing feed-forward networks," *IEEE/ACM Trans. Net.*, vol. 23, no. 5, pp. 1387–1400, 2015.

[40] J. B. Schmitt, N. Gollan, S. Bondorf, and I. Martinovic, "Pay bursts only once holds for (some) non-FIFO systems," in *Proc. of IEEE INFOCOM*, 2011.

[41] Y. Tang, N. Guan, W. Liu, L. T. X. Phan, and W. Yi, "Revisiting GPC and AND connector in real-time calculus," in *Proc. of IEEE RTSS*, 2017.

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

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