Graph-based Deep Learning for Fast and Tight Network Calculus Analyses

Fabien Geyer and Steffen Bondorf

Abstract—Network Calculus (NC) 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. The literature proposes different analyses to consider realistic behavior of networked system such as multiplexing and contention between flows in consecutive queues even though there is no knowledge on the multiplexing discipline employed by the crossed systems (arbitrary multiplexing property). Bounding delays in entire feed-forward networks needs to keep track of such behavior. Moreover, not a single of the existing fast NC heuristics that are based on an algebraic analysis is strictly best. An exhaustive search for the best combination of analyses, i.e., contention modeling, was proposed with the Tandem Matching Analysis (TMA). Additional measures made it scale best among the NC analyses, yet bounding delays may still require several hours of computation time. In this paper, we demonstrate the ability to couple graph-based neural networks with NC by extending TMA with a prediction mechanism replacing the exhaustive search. We propose a framework that learns from NC's TMA, predicts best contention models and feeds them back to TMA where the according NC computations are executed. We achieve provably valid bounds that are very competitive with the exhaustive TMA. We observe a maximum relative error to TMA below 12 %, while execution times remain nearly constant and outperform TMA in differently sized networks by several orders of magnitude.

Index Terms—Deep Learning, Network Calculus.

1 Introduction

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 aims for highest quality of bounds as well as providing a fast analysis that considers all details of the analyzed network[1].

1.1 Problem Overview

In network calculus, a network needs to be modeled by servers (e.g., queues or packet schedulers) whose forwarding capabilities for can be lower bounded. They guarantee an output for their aggregate input of data. Individual data flows traverse sequences of servers where they compete for the forwarding resources with other flows. We do not assume any knowledge about the way data of distinct flows is multiplexed into shared queues at common servers. In NC, this is called arbitrary (or blind) multiplexing. We only assume that the FIFO order of data within individual flows is retained when being multiplexed and forwarded. The data put into a network by a flow is upper bounded in the NC model. This model enables NC to compute deterministic delay bounds.

The NC analysis will compute a bound on an individual flow's end-to-end delay based on such a model. The analyzed flow is commonly known as flow of interest (foi). Under the assumption that no knowledge about the multiplexing of flows is available, the NC analysis must find an internal model of flow contention that *a)* bounds the realistic system's worst-case behavior in the foi's point of view without adding too much pessimism and *b)* can be solved with the capabilities of the available NC analyses. The set of available analyses has been steadily extended in order to capture different features of the modeled network for tightening the derived delay bounds [12, 13, 14, 15]. These alternatives are all proven to result in valid delay bounds for the foi. But among the analyses that can be derived in an algebraic fashion, there is not a single-best one that expresses the realistic worst-case contention model without adding pessimism in some other regard, not even on a tandem of two servers crossed by two flows [16]. Most importantly for our work, we inherit the restriction to a specific shape of curves bounding arrivals and forwarding from [15]: arrivals must be upper-bounded by the minimum of several token-bucket constraints and forwarding service must be lower-bounded by the maximum of several rate-

- *F. Geyer is with Technical University of Munich and Airbus Central Research and Technology, Munich, Germany (email: fabien.geyer@tum.de).*
- *S. Bondorf is with Ruhr University Bochum, Faculty of Mathematics, Bochum, Germany (email: steffen.bondorf@rub.de).*

1. A first version of this work was presented at the 2019 IEEE International Conference on Computer Communications (INFOCOM) [11].

latency constraints. This is, however, a constraint commonly found in a multitude of other NC analyses, too [16, 17, 18].

All the worse for NC, such an algebraic analysis needs to bound the impact of resource contention by transforming the flows' bounding curves between their respective source to the location of contention with the analyzed foi. Curve transformations thus require to backtrack all cross-flows, either in aggregate or separated by worst-case priority assumptions. Different contention models require different flow aggregation/separation assumptions and the resulting structures expressing dependencies of algebraic NC operations become unique. I.e., they all need to be computed.

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) [19]. 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. 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.

In this article, 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. Single backtrackings have been attempted before [14, 15], yet, we are the first to achieve considerably faster execution times than TMA without considerably compromising on delay bound tightness.

1.2 Contributions

While we focus our evaluations on the novel DeepTMA heuristic for NC's TMA, we contribute an entire underlying framework that combines the theories of NC [14] and a graph-based deep learning, namely Graph Neural Networks (GNNs) [20], as well as two of their tools [21, 22]. We assume here feed-forward networks with rate-latency and arbitrary-multiplexing servers, and rate-latency constrained flows, but our approach may be extend to more complex use-cases. DeepTMA achieves the following properties:

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

Our framework does not learn to predict a delay bound but it predicts the most important decisions within the TMA 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.

Fast execution times and high tightness: Recent work [23] about the benefit of technical upscaling showed that TMA cannot be parallelized easily and a speedup of only one order of magnitude was observed. We provide an advancement that improves the execution times of the analysis by multiple order of magnitude.

Limited impact of mismatches between training and application: 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 the involved curve descriptions to learn from, our dataset needs to be restricted in some dimensions. Immediately noticeable is the type of network topologies. We use tandem, sink-tree and random networks for training. An evalutation of the original DeepTMA's performance when applied to other topologies is presented in [24], with a short excerpt in this article.

During the NC network analysis, only one delay bound will be computed – the one for the analyzed flow. The remaining computational effort stems from so-called arrival bounding, the computation of bounds on flow (aggregate) arrivals inside the network. The original DeepTMA was only trained for and applied to minimizing the one delay bound. In this article, we provide an evolution of DeepTMA for arrival bounding while preventing the instantiation and integration of a second, differently trained neural network.

1.3 Outline

The remainder of the article is organized as follows: Section 2 presents the related work on our research direction for network calculus and graph neural networks. Section 3 presents the theory behind our approach in more detail and Section 4 presents our theoretical contribution on combining both areas. In Section 5 we present the combination of tools as well as the generation of a dataset to learn from. Section 6 provides new machine learning-based NC heuristics to benchmark DeepTMA against. These numerical benchmarks are presented in Section 7, followed by observations about the deep learning-based NC heuristics in Section 8. Section 9 concludes our work.

2 Related Work

A recent survey [25] about existing applications of machine learning to formal verification shows that this combination can accelerate formal methods, e.g., theorem proving, model-checking, Boolean satisfiability problems (SAT) or satisfiability modulo theories (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.

2.1 NC 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 [26] or for (max,+) to better fit discrete event systems [27]. Moreover, a common model for NC and event stream theory has been developed [28] and state-based system modeling can be integrated by pairing NC with timed automata [29].

Stochastic extensions to NC were proposed early to deal with, e.g., traffic arrivals following a distribution that cannot be bounded deterministically by an arrival curve. For instance, Boole or martingale inequalities can be applied [30, 31, 32]. This branch of NC was also extended to include statistics and statistical uncertainty to obtain stochastic results [33, 34].

NC has been used to describe component models commonly found in real-time systems [35]. 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 [36, 37, 38].

An optimization formulation has been derived from the NC model that computes tight bounds in networks without assumptions on the multiplexing of flows [17]. 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. There are several algorithms to solve this challenge. As shown in [19], the resulting amount of total orders and therefore linear programs (LP) to solve can quickly becoming prohibitive. [17] proposes a heuristic that skips the expansion step and still derives valid bounds. Its computational demand was numerically evaluated in [19].

Recent works use machine learning to estimate service curves from measurements [39] or to derive traffic characteristics for performing dynamic resource provisioning [40]. 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.

2.2 Deep Learning for Graphs and Formal Verification

GNNs were first introduced in [20, 41], a concept subsequently refined in recent works. Gated Graph Neural Networks (GGNNs) [42] extended this architecture with modern practices by using Gated Recurrent Unit (GRU) memory units [43]. Message-passing neural network were introduced in [44], with the goal of unifying various GNN and graph convolutional concepts. [45] formalized graph attention networks, which enables to learn edge weights of a node neighborhood. Finally, [46] introduced the graph networks (GN) framework, a unified formalization of many concepts applied in GNNs.

These concepts were applied to many domains where problems can be modeled as graphs: chemistry with molecule analysis [47, 44], solving the traveling salesman problem [48], prediction of satisfiability of SAT problems [49], or basic logical reasoning tasks and program verification [42]. For computer networks, they have recently been applied to prediction of average queuing delay [50] and different non-NC-based performance evaluations of networks [22, 51, 52, 53]. In the realm of NC, there is surprisingly little work as of yet. Predating DeepTMA [11] we base our work on, there is an effort to predict the delay bound computed by different NC analyses by using GNNs. Each of these analyses only considers a pre-defined contention model whenever there are alternatives for a tandem. The prediction is then used to only execute the most promising analysis [54].

3 Background

3.1 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 [20, 41].

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 $\mathbf{i}_v \in \mathbb{R}^n$ and $\mathbf{o}_v \in R^m$ represent respectively the input features (e.g. node type, service or arrival curve parameters) and output values for node v (e.g. decision for the NC analysis). The concept behind GNNs is called *message passing*, where so-called hidden representations of nodes $\mathbf{h}_v \in \mathbb{R}^k$ are iteratively passed between neighboring nodes. Those hidden representations are propagated throughout the graph using multiple iterations until a fixed point is found or after a fixed number of iterations. 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 iteration 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 [41], such that:

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

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) [42] 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) [43] 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 1.

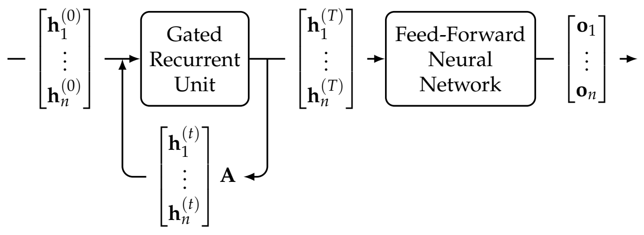


Figure 1: Gated Graph Neural Network architecture.

Formally, the propagation of the hidden representations $\mathbf{H}^{(t)}$ among neighboring nodes for one time-step is formulated as:

$$\mathbf{H}^{(t)} = \left[\mathbf{h}_1^{(t)}, \ldots, \mathbf{h}_{|\mathcal{V}|}^{(t)}\right] \tag{5}$$

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

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

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

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

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

where $\sigma(x) = 1/(1 + e^{-x})$ is the logistic sigmoid function and $\odot$ 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 adjacency matrix, determining the edges in the graph $\mathcal{G}$.

Equation (6) 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 (7) to (10) correspond to the mathematical formulation of a GRU cell [43], with Equation (7) representing the GRU reset gate vector, Equation (8) the GRU update gate vector, and Equation (10) the GRU output.

In order to propagate the hidden representations throughout the complete graph, a fixed number of iterations of Equations (7) to (10) are performed. This extension has been shown to outperform standard GNN which require to run the recursion until a fixed point is found.

We also extended our neural network architecture with an edge attention mechanism similar to the one proposed in [45]. 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 $\mathbf{h}_u^{(t)}$:

$$\rho_{v,u}^{(t)} = \sigma\left(\mathbf{W}_a\left\{\mathbf{h}_v^{(t)}, \mathbf{h}_u^{(t)}\right\} + \mathbf{b}_a\right) \tag{11}$$

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

$$\mathbf{h}_v^{(t)} = \sum_{u \in \textsc{Nbr}(v)} \rho_{v,u}^{(t-1)} f\left(\mathbf{h}_u^{(t-1)}\right). \tag{12}$$

3.2 Network Calculus

The NC model of a network is a directed graph called server graph $\mathcal{G}_{NC} = (\mathcal{V}_{NC}, \mathcal{E}_{NC}, \mathcal{F})$ with servers $s \in \mathcal{V}_{NC}$, edges $(v, u) \in \mathcal{E}_{NC}$ and data flows $f \in \mathcal{F}$. Servers represent the forwarding locations in a network, e.g., queues or packet schedulers. They guarantee a lower bound on data forwarding and thus an output quantity given an input quantity of data. Flows travel along directed edges, crossing servers and demanding their forwarding service. I.e., they define the input quantity of servers. NC models this with an upper bound on the flow's data arrivals valid in any duration of time. Figure 2 shows a server graph, a tandem.

Definition 1 (Tandem of Servers). A server graph $\mathcal{T} = (\mathcal{V}_{NC}, \mathcal{E}_{NC}, \mathcal{F})$ is called a tandem if the following properties hold: $|\mathcal{E}_{NC}| = |\mathcal{V}_{NC}| - 1$. For any server $s \in \mathcal{V}_{NC}$, let $in(s)$ be the amount of directed edges ending in server s and $out(s)$ be the amount of edges starting in s. On a tandem, it holds $\forall s \in \mathcal{V}_{NC} \mid \max(in(s)) = \min(1, \max(out(s))) = 1$.

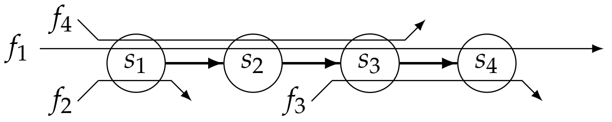


Figure 2: Server Graph Model in NC.

In this paper, we put some additional assumptions on the NC model:

- There cannot be cyclic dependencies between flows. This is achieved by a restriction to feed-forward networks such as the tandem network of Figure 2. Any network can be converted to a feed-forward one [55].
- When multiple flows multiplex at a server, e.g., f_1, f_2 and f_4 at s_1 in Figure 2, we do not know the resulting order of their data. This is called arbitrary multiplexing.
- However, we assume that the order of data within a single flow will not change by multiplexing or forwarding.
- Flows are routed along point-to-point paths.
- The NC curves that bound data arrivals and forwarding capabilities are restricted to certain shapes: the minimum over multiple token buckets (like IntServ's TSPEC) and the maximum over multiple rate latencies, respectively. Details can be found in [16, 17, 18].

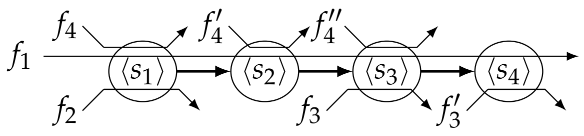
A network calculus analysis takes such a model as the input and computes a bound on a specific flow's end-to-end delay – as close to the realistic worst case as possible – with the least computational effort possible. The analyzed flow is called flow of interest (foi) and the set of tandem analyses has been steadily extended in order to improve tightness of the foi's delay bound. Two leaps have been taken by incorporating the Pay Bursts Only Once (PBOO) [14] and the Pay Multiplexing Only Once (PMOO) [15] properties into the NC analysis. Both mitigate previously added pessimism not found in a realistic system but required by the analysis. PBOO prevents the bound on the foi's worst-case burstiness to appear multiple times in the analysis as if it built up at every server – an unrealistic contention model. PMOO extends this to the burstiness of cross-traffic present on consecutive servers on the foi's path. In NC's interpretation as term rewriting [56], these two analyses are reduction rules. Their central means of transforming tandems is *cutting*.

Definition 2 (Cutting NC Tandems). Given a tandem $\mathcal{T} = (\mathcal{V}_{NC}, \mathcal{E}_{NC}, \mathcal{F})$ and a NC analysis $\mathcal{A}$, a cut marks edge $e \in \mathcal{E}_{NC}$ such that $\mathcal{A}$ will analyze $\mathcal{T}$ as a sequence of sub-tandems $\langle \mathcal{T}_l, \mathcal{T}_r \rangle$ where $\mathcal{T}_l$ holds all the model information to the left of e and $\mathcal{T}_r$ that to the right of e. A cutting (also called combination of cuts) is a set of cuts on $\mathcal{T}$.

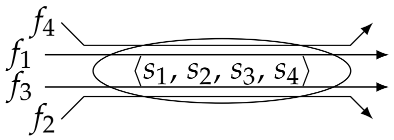
Visually, the analyses implementing PBOO (Seperate Flow Analysis, SFA [14]) and PMOO (PMOO analysis, PMOOA [15]) proceed as depicted in Figure 3a and 3b:

All Cuts (Figure 3a): SFA cuts every edge in the NC model along with the flows crossing it (except the foi f_1). The resulting sub-tandems are demarcated with $\langle\cdot\rangle$ and consist of single servers. For the cut flows, their arrivals at the subsequent server need to be bounded (we denoted the respective location with f_i'). Such a flow's bound consists of its initial – given burstiness – bound plus the worst-case increase due to having crossed the previous servers.

No Cuts (Figure 3b): Without cuts, the entire tandem is analyzed at once. Mitigating the need for deriving bounds on flow arrivals in the network allows for achieving the PMOO property in addition to PBOO. For details on how to implement a no-cuts analysis, we refer the reader to [15].



(a) All Cuts achieves the PBOO property.



(b) No Cuts achieves the PMOO property.

Figure 3: Reduction rules in the NC analysis.

The two reduction rules are part of the algebraic NC analysis branch. It was discovered that the algebraic analysis pays for its ability to apply the no-cuts reduction with the loss of server order information. As a consequence, neither of the reduction rules are generally resulting in a tighter delay bound than the other one. Therefore, the optimization-based NC analysis branch was proposed [16]. Later, a tight optimization analysis was developed [17], along with a heuristic that promises better scalability with increasing network size. In [19], it was shown that said heuristic may not scale well and that it is rivaled by algebraic NC in terms of delay bound tightness, too. The so-called Tandem Matching Analysis (TMA) we base our work on partially overcomes the computational effort challenges imposed by the exhaustive search over all combinations of the reduction rules above. See Figure 4 for two alternative rules to *all cuts* and *no cuts*. With DeepTMA, we make the approach scale even better by predicting the best tandem matching, i.e., combination of cuts, instead of exhaustively searching for it.

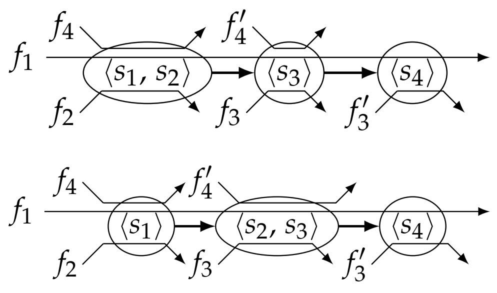


Figure 4: Some additional reduction rules applied by TMA.

4 Graph Neural Network for NC

We develop our DeepTMA heuristic in this section. It is based on the concept of GNN introduced in earlier. The goal of DeepTMA is to predict the best tandem decompositions, i.e., combinations of cuts, to use in TMA. For simplicity, we refer to NC server graphs as networks and to the graph model used in GNN as graphs.

The main intuition is to transform the NC server graph and flows into an undirected graph. This graph representation is then used as input for a neural network architecture able to process general graphs, which will then predict the tandem decomposition resulting in the best residual service curves. Our approach is illustrated in Figure 5. Since the delay bounds are still computed using the formal network calculus analysis, they inherit their provable correctness.

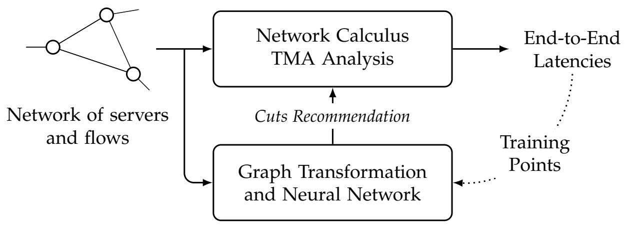

Network of servers and flows
Network Calculus TMA Analysis
End-to-End Latencies
Cuts Recommendation
Training Points
Graph Transformation and Neural Network

Figure 5: Overview of the proposed approach.

4.1 Application to TMA

In order to apply the concepts described in Section 3.1 to a network calculus analysis, we model NC's directed network as an undirected graph. Figure 6 illustrates this graph encoding on the network from Figure 2.

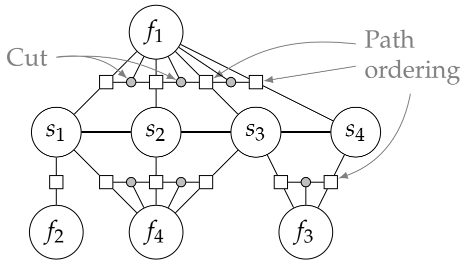


Figure 6: Transformed network of Figure 2 to the graph model.

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 containing the hop count as feature 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 are as follows:

- For each server s, parameters of its rate-latency 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 token-bucket 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 an integer: $PathOrder$
- Finally, neither cut nodes nor edges have input features

Equation (13) illustrates the matrix encoding of part of the graph from Figure 6.

$$\begin{array}{ccccccccc|c}
\text{Server} & \text{Flow} & \text{Path Order} & \text{Cut} & \text{S. Rate} & \text{S. Latency} & \text{F. Rate} & \text{F. Burst} & \text{Hop count} & \\
1 & 0 & 0 & 0 & R_{s_i} & L_{s_i} & 0 & 0 & 0 & s_i \\
0 & 1 & 0 & 0 & 0 & 0 & r_{f_k} & b_{f_k} & 0 & f_k \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & P_{f_1,s_1} \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 2 & P_{f_1,s_2} \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 3 & P_{f_1,s_3} \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 4 & P_{f_1,s_4} \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & C^{f_k}_{s_i,s_j}
\end{array} \tag{13}$$

Note, that the above restriction to (single) rate-latency curves for the service capabilities and (single) token-bucket curves for arrival constraints is not a restriction of our DeepTMA analysis. It is trivial to extend input features to the larger set of curve parameters required to model the curve shapes mentioned in Section 3.2.

Based on this description of the server graph, the problem of choosing the best tandem decomposition to give to the NC analysis is formulated 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 binary cross-entropy loss function is used during training for this classification problem. The other nodes of the graph are masked from the loss function.

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. The prediction of the best decomposition for a given tandem, starting with the foi's path, is done by iterating over all potential cuts and selecting the ones which have been classified as cutting points for said tandem.

4.2 Best Contention Models Across the Entire Analysis

Figures 3 and 4 in Section 3.2 already show the need for arrival bounding on the foi's path – see flow labels f'_i and f''_i. Moreover, our sample tandem assumes that bounds on flow arrivals are known when entering the tandem. This need not be the case in a feed-forward network where cross-flows traversed multiple servers before interfering with the flow of interest. Therefore cross-flow arrivals are required to be computed here, too. Bounding the arrivals of cross-traffic becomes a resource intensive, recursive procedure [57, 5]. It starts with the foi and it only terminates in feed-forward networks when all cross-flows are backtracked to their sources. The procedure is visualized in Figure 7.

Applying the exhaustive TMA in every recursion level (i.e., every cycle in Figure 7) yields large computational demands. Given a tandem of length n servers, TMA tests all $2^{(n-1)}$ combinations of cuts. Visually, TMA unwinds all loops and branches (see dashed line) that can be taken in the cycle. On the other hand, the minimum-cost NC analysis can be obtained by unwinding only the bare minimum of loops: do not take optional branches by choosing a single contention model like PBOO (SFA) or PMOO do. With DeepTMA, we create an alternative heuristic that is not deciding on the cut combination in a static way. Instead, our GNN uses a range of input features to predict the best cut combination.

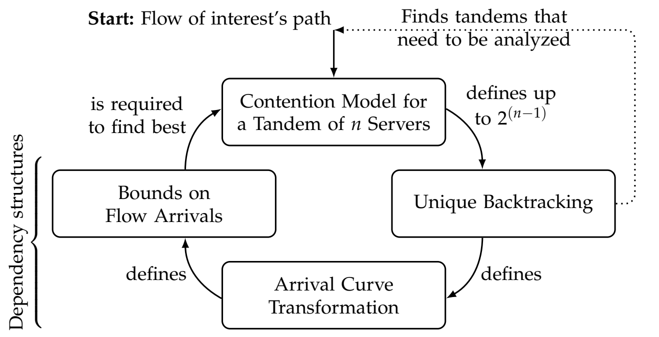

Start: Flow of interest's path
Finds tandems that need to be analyzed
Contention Model for a Tandem of n Servers
is required to find best
defines up to $2^{(n-1)}$
Bounds on Flow Arrivals
Unique Backtracking
defines
Arrival Curve Transformation
defines
Dependency structures

Figure 7: Dependency cycle defining current NC analyses.

The original DeepTMA of [11] was designed to predict the best cuts for a tandem of servers, subject to minimizing the analyzed flow's delay bound. However, the delay bound is only computed in the very first iteration of Figure 7's loop (after "**Start:** Flow of interest's path"). As seen above, any subsequent iteration will be part of the arrival bounding of cross flows. In this article, we evolve DeepTMA to also provide delay-bound-minimizing cut combinations for the bounding cross-traffic arrivals. Our delay-bound-based approach has the advantage of not instantiating, training and integrating a second GNN for the purpose of arrival bounding. Figure 8 shows that our approach is indeed superior in the vast amount of cases and we will apply this DeepTMA-based arrival bounding in all our evaluations.

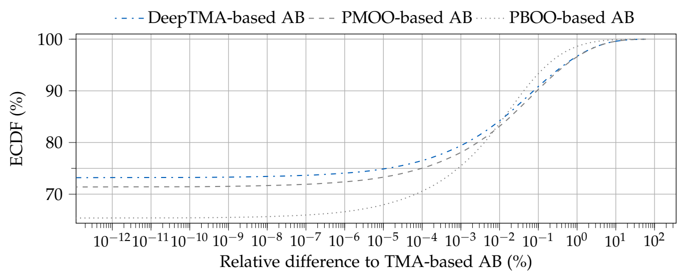


Figure 8: Comparison of arrival bounding (AB) methods on the dataset constructed in Section 5.1.

5 Implementation and Dataset Generation

We implemented DeepTMA and the graph neural network architecture using PyTorch [58]. The recursion from Equation (1) was dynamically unrolled for a fixed number of iteration according to the diameters of the analyzed graphs. Table 1 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:	209 764 parameters

Table 1: Size of the layers used in the GGNN. Indexes represent respectively the weights (w) and biases (b) matrices.

We analyzed each network with the NetworkCalculus.org Deterministic Network Calculator (NCorg DNC)[2] version 2.6.0 and perform the exhaustive TMA analysis to generate the best cuts combinations. 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.

5.1 Dataset Generation

In order to train our neural network architecture, we follow a traditional supervised learning approach. We randomly generated a set of random topologies according to three different random topology generators: *a)* tandems or daisy-chains like in Figure 2, *b)* trees and *c)* random server graphs following the Erdős–Rényi model [59], then made feed-forward with the Turn Prohibition algorithm [55]. For each created server, a rate latency service curve was generated with rate and latency parameters taken from a uniform distribution. A random number of flows with random source and sink servers was added. For each flow, a token bucket arrival curve was generated with burst and rate parameters taken from a uniform distribution. All curve parameters were normalized to the $(0, 1]$ interval. In total, 172 374 different networks were generated, with a total of more than 13 million flows, and close to 260 million tandem decompositions. Half of the networks were used for training the neural network, while the other half was used for the evaluation presented later in Section 7. Table 2 summarizes different statistics about the generated dataset. The dataset is available online[3] to reproduce our learning results.

Parameter	Min	Max	Mean	Median
# of servers	2	41	14.6	12
# of flows	3	203	101.2	100
# of tandem combinations	2	197 196	1508.5	384
# of nodes in analyzed graph	10	2093	545.2	504
# of tandem combination per flow	2	65 536	19.4	4
# of flows per server	1	173	18.1	10

Table 2: Statistics about the generated dataset.

6 Other TMA Heuristics

To benchmark DeepTMA, we present three additional heuristics for the choice of TMA's tandem decompositions. Compared to the GNN-based proposal, those heuristics are based on simpler algorithms.

2. Formerly known as DiscoDNC [21], see networkcalculus.org/dnc
3. `https://github.com/fabgeyer/dataset-deeptma-extension`

6.1 RND: Random Choice of Tandem Decomposition

The simplest heuristic is to randomly select multiple alternative tandem decompositions, where each decomposition has the same probability of being chosen. Given any n-server tandem, starting with the foi's path as shown in Figure 7, 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.

6.2 Simplified Machine Learning Heuristics

While DeepTMA is based on an approach which uses the complete information about the server graph, we propose here a simpler machine-learning approach which uses a simplified view of the server graph and its features. As for DeepTMA, this heuristic uses machine learning algorithms in order to classify each cut in the same two classes, namely decide to perform a cut between a pair of servers or not.

This simplified approach only uses a local view of the network, i.e. parameters of the pair of servers between which the cut is located, named here *source* and *sink*. For each cut in the network, we define a feature vector comprised of the following values:

- *FlowArrival{Rate,Burst}*: the parameters of the token bucket arrival curve of the foi;
- *FlowPathLen*: the path length of the foi;
- *CutOrder*: the index of the cut in the path of the foi;
- *{Source,Sink}Service{Rate,Latency}*: the parameters of the rate-latency service curves of the pair of servers;
- *{Source,Sink}SumArrival{Rate,Burst}*: the sum of the parameters of the arrival curves of the flows traversing each server of the pair;
- *SourceNFlows* and *SinkNFlows*: the number of flows at each server of the pair.

While this simplified view of the server graph performs worse than the one used DeepTMA – as show later in Section 7 – our main motivations for this simplified approach are simplicity and explainability of the model. Feature importance [60, 61] is more easily performed on such simplified feature vector than on the GNN model, as illustrated later in Section 8.

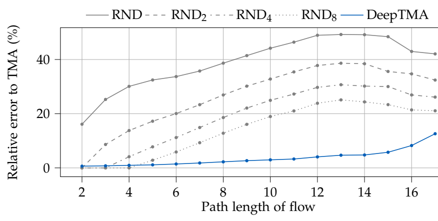
Using those feature vectors, we propose here two heuristics, one based on feed-forward neural network, and one based on random forests. Since the output of both heuristics is a probability of making a cut, multiple tandem decompositions can be generated using the approach presented later in Section 6.3 and Algorithm 1.

6.2.1 FFNN: Feed-Forward Neural Network Heuristic

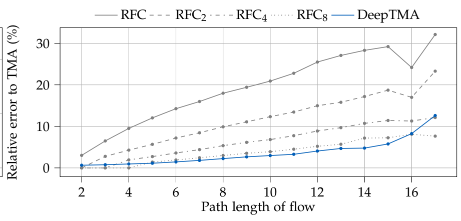
This heuristic uses a standard multi-layer feed-forward neural network to classify the cuts using the simplified feature vector presented earlier. The size and number of hidden layers of the FFNN is detailed in Table 3. We use standard training method based on gradient descent to train the neural network. As for DeepTMA, our implementation is based on PyTorch [58].

6.2.2 RFC: Random Forest Classifier Heuristic

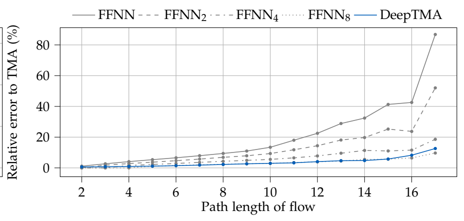
This heuristic uses random forests [60] to classify the cuts using the simplified feature vector presented earlier. Our implementation of this heuristic is based on scikit-learn [62].



(a) Comparison against RND heuristic.



(b) Comparison against RFC heuristic.



(c) Comparison against FFNN heuristic.

Figure 10: Relative error of DeepTMA and the heuristics presented in Section 6.

Layer	**Size**
input	$(10, 64)_w + (64)_b$
hidden layers	$2 \times \{(64, 64)_w + (64)_b\}$
out final	$(64, 2)_w + (2)_b$
Total:	9154 parameters

Table 3: Size of the layers used in the FFNN. Indexes represent respectively the weights (w) and biases (b) matrices.

6.3 Generating multiple decompositions

Given a foi and a cut, the output of the machine learning-based heuristics presented earlier is a probability of cutting. This probability is generated by the neural networks using the softmax function after the last layer. In case a single tandem decomposition has to be generated, the decision of cutting or not is made using a threshold of 50 %.

The cut probabilities may also be used in order to generate multiple tandem decompositions as illustrated in Algorithm 1. In case the number of tandem decompositions is lower than the number of requested decompositions, we simply return all combinations of cuts. Otherwise, we sample the distribution of cuts in order to generate the decompositions. In Section 7, we label those extended heuristics using n as subscript, with n the number of decompositions.

Algorithm 1 Generation of n tandem decompositions for a flow traversing $L + 1$ servers.

```
if $n \leq 2^L$ then return all combinations of cuts
else
    for all $i := 1$ to $n$ do
        $v \leftarrow [c_1, \ldots, c_L] \sim \mathcal{U}(0,1)^L$
        cuts$_i \leftarrow \mathbb{I}\left(v \leq \left[\Pr(cut^{GNN}_{foi,1}), \ldots, \Pr(cut^{GNN}_{foi,L})\right]\right)$
            ($\mathbb{I}$ is the indicator function)
    return {cuts$_1, \ldots,$ cuts$_n$}
```

7 Numerical Benchmarks

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

Unless specified otherwise, all the evaluations presented in this section were performed on the dataset described in Section 5.1. In order to perform the evaluation, the dataset was split in two parts: one part was used for training the machine learning-based heuristics, while the second part was used to perform the numerical evaluations presented in this section. Additionally, DeepTMA was also evaluated on the set of network from [19] in Section 7.2, and on the set of networks from [11] in Section 7.4.

7.1 Relative Error

We investigate in this section the resulting loss of tightness in case a non-optimal tandem decomposition was selected by a given heuristic. In order to quantitatively evaluate this loss of tightness compared to TMA, we use the relative error, defined as:

$$RelErr_{\text{foi}} = (delay^{\text{heuristic}}_{\text{foi}} - delay^{\text{TMA}}_{\text{foi}}) / delay^{\text{TMA}}_{\text{foi}} \quad (14)$$

Classical NC Analyses

Figure 9 illustrates the relative error of DeepTMA against classical NC analyses. DeepTMA-derived delay bounds are tightest among these heuristics, deviating from TMA by no more than 12 % in our experiments in the worst-case.

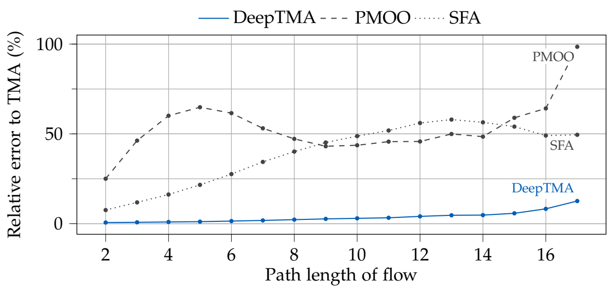


Figure 9: Relative error of DeepTMA and existing NC heuristics.

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 benchmarks against newly contributed ML-based heuristics for TMA.

New Heuristics

Figure 10 compares DeepTMA against the other heuristics introduced in Section 6. Only FFNN$_8$ and RFC$_8$ are able to achieve a relative error similarly small as DeepTMA on the larger networks, yet, at a much larger computational cost since 8 different tandem combinations and their entire dependency structures have to be evaluated every time.

7.2 Scalability and robustness on larger networks

Additionally to the dataset which was presented in Section 5.1, we also evaluate our approach on the set of networks used in [19]. No additional training of the GNN is performed on this additional dataset. Table 4 summarizes different statistics about this additional dataset. Compared to dataset used for training, this additional set of networks is up to two order of magnitude larger in term of number of servers and flows per network. We evaluate here if our approach is able to scale to such larger networks in terms of tightness.

Parameter	Min	Max	Mean	Median
# of servers	38	3626	863.0	693
# of flows	152	14 504	3452.0	2772
# of tandem combinations	2418	121 860	24 777.6	18 869
# of nodes in analyzed graph	1358	113 162	25 137.7	19 518
# of tandem combination per flow	2	512	7.3	8
# of flows per server	1	467	16.4	12

Table 4: Statistics about the set of networks from [19].

Figure 11 illustrates the relative error of DeepTMA compared to a random heuristic which selects the tandem decompositions randomly. DeepTMA achieves relative errors that are two orders of magnitudes smaller than the random heuristics, resulting in better end-to-end delay bound accuracy w.r.t. the exhaustive TMA. Although DeepTMA was not trained on such large networks, the relative error still stays below 0.3 % even on the larger networks. Those results highlight that DeepTMA is indeed able to scale to networks much larger than to those it was initially trained for.

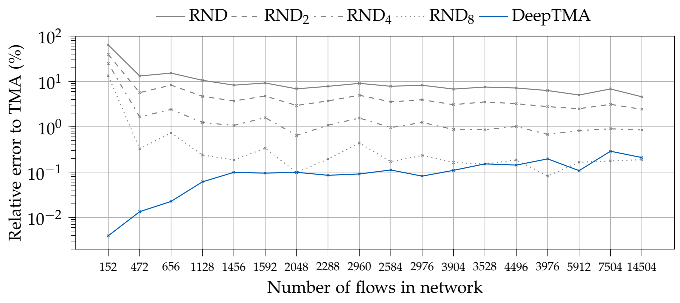


Figure 11: Relavitve error of DeepTMA on the dataset from [19] with much larger networks.

Additional results regarding robustness of DeepTMA with respect to scalability on larger networks can also be found in [24].

7.3 Training time

We illustrate in Figure 12 the evolution of the relative error during the training phase of the GNN. As noted in Section 5.1, this training was done 86 187 topologies. Training duration was measured while training was done using a Nvidia GTX 1080 Ti GPU.

7.4 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.

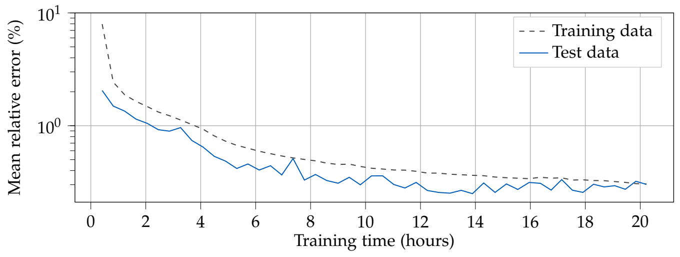


Figure 12: Evolution of the relative error during training.

Classical NC Analyses

Figure 13 shows benchmarking results of DeepTMA against the classical analysis in NC. We compare against TMA and the established SFA [14] and PMOO [15] 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. This implies two things: first, the overhead of querying for predictions is not necessarily large and secondly, the contention model tends be closer to PMOO than to SFA, consisting of tandems of multiple servers.

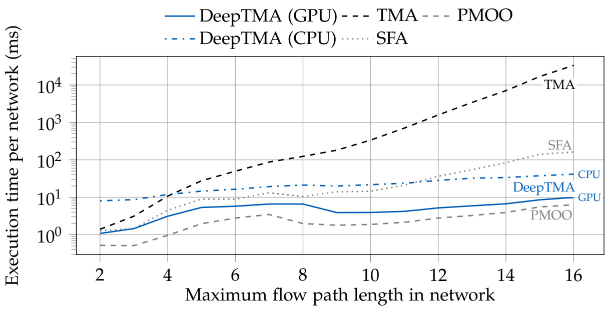


Figure 13: Comparing DeepTMA to existing NC heuristics on the dataset from [11].

TMA

Since DeepTMA can be executed on either CPU or GPU, we first compare both platforms and their affinity at parallelization in Figure 14. 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 (6) 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 4 is automatically performed by PyTorch. We present in Figure 14 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.

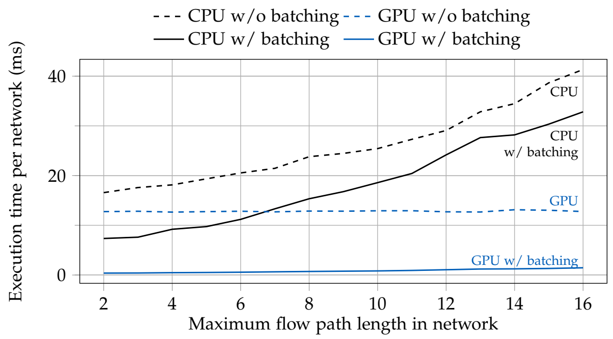


Figure 14: Execution time of the cut recommendation part of DeepTMA, executed on CPU or GPU, without batching or batch sizes of 64 networks on the dataset from [11].

In addition, we measured the execution time of TMA using the NCorg DNC [21]. The same CPU was used for running NCorg DNC, with Oracle's HotSpot JVM version 1.8.

Whereas Figure 14 provides insight on the computational cost of DeepTMA, Figure 15 compares it to a generalized version of the heuristics presented in Section 6. 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 NCorg DNC 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 Section 7.1 that RND_i $i \in 4, 8$ is 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.

7.5 Memory footprint

We evaluate in Figure 16 the memory footprint of the classical TMA against the GNN heuristic. Compared to the classical NC analysis, the GNN requires almost an order of magnitude less memory, even on the larger networks with up to 14 504 flows. In summary, those results and the

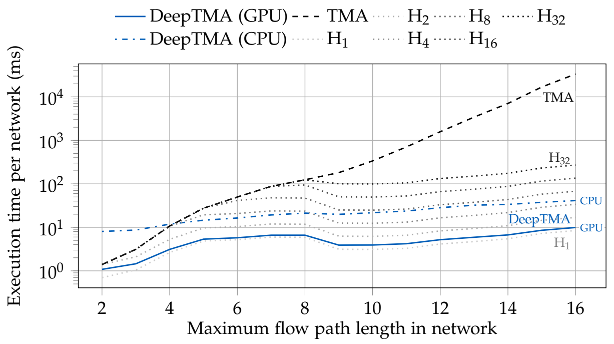


Figure 15: Execution times per topology for TMA, DeepTMA and H_n heuristics on the dataset from [11].

results from Section 7.4 illustrate that DeepTMA requires only a fraction of the computing and memory resources of the classical analysis, with only a minor drop in tightness.

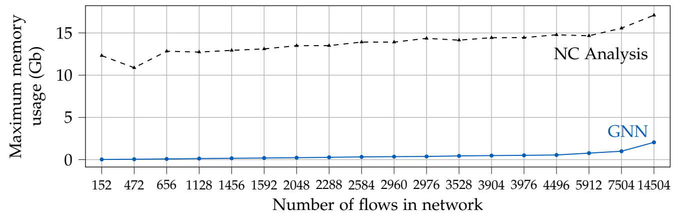


Figure 16: Memory usage of the GNN and the TMA NC analysis on the dataset from [11].

8 Insights into Learning Approaches

In order to better understand the importance of the input features used in DeepTMA and the two other machine learning-based heuristics proposed in this article, we perform in this section a sensitivity analysis of TMA and assess DeepTMA's features' importance.

8.1 Sensitivity analysis

We perform here a sensitivity analysis of TMA's cut choices in order to better understand which parameters influence the choice of best cuts. To numerically evaluate the sensitivity, we randomly modify the service and arrival curve parameters $p_{original}$ of our evaluation networks with a relative scale s according to the following uniform distribution:

$$p_{new} \sim \mathcal{U}\left(p_{original}(1-s), p_{original}(1+s)\right) \quad (15)$$

We then compare the share of flows where best cuts have changed due to the random change of curves parameters. Results are presented in Figure 17.

The service rate parameter influences the most the choice of cuts, where even minimal changes result in different choices. Arrival curve parameters also impact also TMA's analysis, but with less magnitude than the service rate. Finally, the service latency has almost no influence on the choice of cuts, where even large changes of its value result in less than 1 % of changed choices.

These observations can be explained by the fundamental impact a cut in TMA has on the composition of sub-tandem residual service that is composed to an end-to-end guaranteed service for the foi. On any tandem, forwarding is lower bounded by the minimum residual rate over all servers. Separating a subset of faster or slower servers by cutting can therefore easily enable to make use of larger residual service on separated sub-tandems, in particular when bounding cross-flow arrivals. The service latency, in contrast, is an additive factor in the residual forwarding service computation, making it considerably less impactful. Cutting a link traversed by a cross-flow means computing an arrival curve for said flow at this location, an output bound after having traversed the previous sub-tandem. These output bounds consist of the original burstiness and a burstiness increase due to queueing. Thus, a significant change in the burstiness increase is required to change the choice of cut set. This is less likely to be achieved by solely modifying original arrival curves' rate or burstiness parameters.

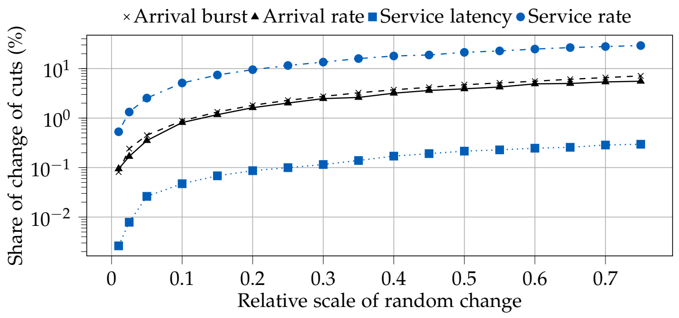


Figure 17: Sensitivity analysis of the TMA cut choices.

8.2 Feature Importance

We use the permutation-based importance measure [60, 61] in order to assess each feature's importance of DeepTMA. For each input feature presented in Sections 4.1 and 6.2, we randomize it by randomly permuting its values in the training set, and assess the impact it has on the relative error of the predictions. We define the importance metric as:

$$Importance(Feature) = \frac{1}{|\mathcal{F}|}\sum_{f_i\in\mathcal{F}}\left(RelErr_{f_i}^{Feature} - RelErr_{f_i}^{Baseline}\right) \quad (16)$$

with the baseline corresponding to the method without any feature permutation. With this evaluation, we assess how much the GNN model relies on a given feature of interest for making its prediction.

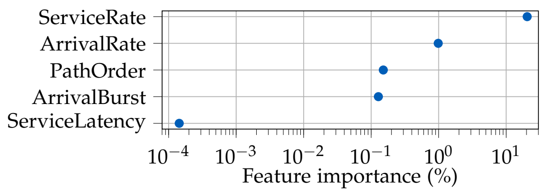


Figure 18: Feature importance according to feature permutation method for DeepTMA.

DeepTMA's features importance are presented in Figure 18. The service rate of the servers in the network have the largest impact on the final decision of cutting. The remaining features appear to have less importance on the cut prediction, with almost no importance for the service latency. These results confirm the findings presented in Figure 17, showing that the GNN matches TMA's sensitivity. Figure 18 also highlights that the arrival rate is of considerably larger importance than the arrival burstiness whereas the sensitivity analysis showed slightly more impact of a changing burstiness. The importance of the arrival rates is natural to NC with arbitrary multiplexing: the derived worst-case assumption is that the flow of interest is served with the residual forwarding capabilities and the according service curve's latency increases fast with increasing utilization. Thus, the (cross-traffic) arrival rates must be an important feature.

Interesting from the NC perspective is the observation that the order of servers (PathOrder) has a percental importance two orders of magnitude lower than the service rate. In combination, these two features constitute the very reason for TMA (and optimization-based analyses, see [16]) to outperform the previous NC analyses.

We also assess the importance of other flows and other servers on a cut. We perform this by assessing the number of iterations of message passing (i.e. Equation (1)) and the impact it has on the relative error. As for feature importance, we compare the results according to Equation (16). Results are presented in Figure 19. The first 4 loop iterations appear to have the largest impact on the cut decision, meaning that the cut decision is mainly based on information from servers close to the cut. We notice that the importance drops sharply after 5 iterations, and converges after 15 iterations. This indicates that servers and flows farther away from the cut decision are less relevant to the cut decision – an insight to potential further improvement of DeepTMA's tradeoff between computational effort and relative error.

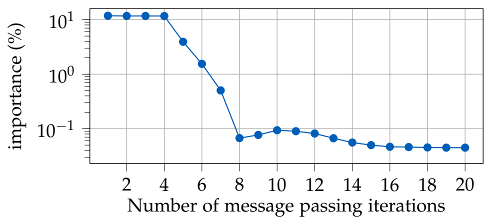


Figure 19: Importance of message passing iterations.

We evaluate also the features importance of the FFNN heuristic in Figure 20. Overall, those results confirm the ones presented for DeepTMA, namely that the service rate of the servers have the largest impact on the cutting decision. Interestingly, the impact of the other flows is more related to the aggregated arrival burst (*SourceSumArrivalBurst* and *SinkSumArrivalBurst*) than the aggregated arrival rate compared to DeepTMA. Finally, as for DeepTMA, the position of the cut in the path of the flow is not relevant.

Finally, we also evaluate the features importance of the RFC heuristic in Figure 21. The results for the RFC heuristic are in line with the ones from DeepTMA and exhibit as similar ranking than for the FFNN. Overall, Figures 18, 20 and 21 confirm existing results of the NC analysis presented in Section 8.1, namely its sensitivity to service rate and the importance of other flows.

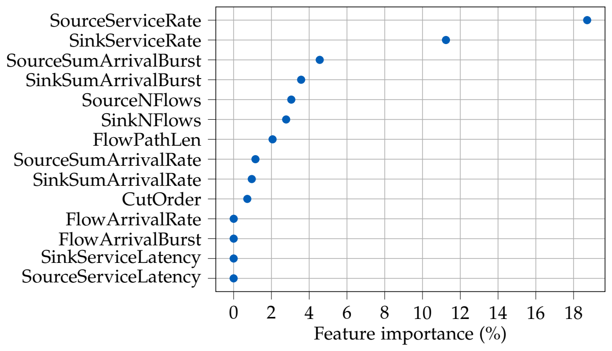


Figure 20: Feature importance according to feature permutation method for the FFNN heuristic.

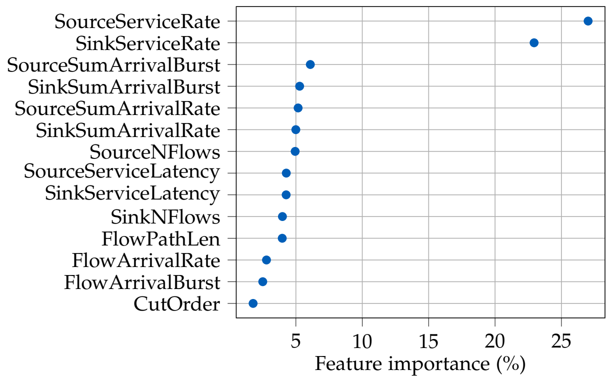


Figure 21: Feature importance according to feature permutation method for the random forests classifier.

9 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. Via a numerical evaluation, we show that our heuristic is accurate and produces end-to-end delay bounds which are almost as tight as TMA, with an execution time several orders of magnitude smaller than TMA and a memory footprint an order of magnitude smaller. 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. Numerical evaluations on large networks with up to 14 000 flows also illustrate that our approach is able to scale despite having being trained on much smaller networks.

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. Our method outperforms simpler ML-based methods, justifying the use of a more complex machine learning method. Finally some insights into the learning is also given via an evaluation of feature importance, confirming existing results of NC.

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, 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, 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] F. Geyer and S. Bondorf, "DeepTMA: Predicting effective contention models for network calculus using graph neural networks," in *Proc. of INFOCOM*, 2019.
[12] R. L. Cruz, "A calculus for network delay, part I: Network elements in isolation," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, 1991.
[13] "A calculus for network delay, part II: Network analysis," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, 1991.
[14] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, 2001.
[15] 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.
[16] J. B. Schmitt, F. A. Zdarsky, and M. Fidler, "Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch. . . ," in *Proc. of IEEE INFOCOM*, 2008.
[17] 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.
[18] A. Bouillard and G. Stea, "Exact worst-case delay in FIFO-multiplexing feed-forward networks," *IEEE/ACM Trans. Net.*, vol. 23, no. 5, 2015.
[19] 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, 2017.
[20] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proc. of IEEE IJCNN*, 2005.
[21] S. Bondorf and J. B. Schmitt, "The DiscoDNC v2 – a comprehensive tool for deterministic network calculus," in *Proc. of EAI ValueTools*, 2014.
[22] F. Geyer, "Performance evaluation of network topologies using graph-based deep learning," in *Proc. of EAI ValueTools*, 2017.
[23] A. Scheffler, M. Fögen, and S. Bondorf, "The deterministic network calculus analysis: Reliability insights and performance improvements," in *Proc. of IEEE CAMAD*, 2018.
[24] F. Geyer and S. Bondorf, "On the Robustness of Deep Learning-predicted Contention Models for Network Calculus," in *Proc. of IEEE ISCC*, 2020.
[25] M. Amrani, L. Lúcio, and A. Bibal, "ML + FV = ♡? A survey on the application of machine learning to formal verification," 2018, arxiv:1806.03600.
[26] H. Al-Zubaidy, J. Liebeherr, and A. Burchard, "A (min, ×) network calculus for multi-hop fading channels," in *Proc. of IEEE INFOCOM*, 2013.
[27] J. Liebeherr, "Duality of the max-plus and min-plus network calculus," *Found. Trends. Network.*, vol. 11, no. 3-4, 2017.
[28] M. Boyer and P. Roux, "Embedding network calculus and event

stream theory in a common model," in *Proc. of IEEE ETFA*, 2016.
[29] 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.
[30] C.-S. Chang, *Performance Guarantees in Communication Networks*. Springer, 2000.
[31] F. Ciucu, A. Burchard, and J. Liebeherr, "A network service curve approach for the stochastic analysis of networks," in *Proc. of ACM SIGMETRICS*, 2005.
[32] F. Ciucu, F. Poloczek, and J. B. Schmitt, "Sharp per-flow delay bounds for bursty arrivals: The case of FIFO, SP, and EDF scheduling," in *Proc. of IEEE INFOCOM*, 2014.
[33] M. A. Beck, S. A. Henningsen, S. B. Birnbach, and J. B. Schmitt, "Towards a statistical network calculus – dealing with uncertainty in arrivals," in *Proc. of IEEE INFOCOM*, 2014.
[34] F. Dong, K. Wu, and V. Srinivasan, "Copula analysis for statistical network calculus," in *Proc. of IEEE INFOCOM*, 2015.
[35] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Proc. of ISCAS*, 2000.
[36] N. Guan and W. Yi, "Finitary real-time calculus: Efficient performance analysis of distributed embedded systems," in *Proc. of IEEE RTSS*, 2013.
[37] K. Lampka, S. Bondorf, and J. B. Schmitt, "Achieving efficiency without sacrificing model accuracy: Network calculus on compact domains," in *Proc. of IEEE MASCOTS*, 2016.
[38] K. Lampka, S. Bondorf, J. B. Schmitt, N. Guan, and W. Yi, "Generalized finitary real-time calculus," in *Proc. of IEEE INFOCOM*, 2017.
[39] S. K. Khangura, M. Fidler, and B. Rosenhahn, "Neural networks for measurement-based bandwidth estimation," in *Proc. of IFIP Networking*, 2018.
[40] Q. Xu, J. Wang, and K. Wu, "Learning-based dynamic resource provisioning for network slicing with ensured end-to-end performance bound," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 1, 2020.
[41] 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, 2009.
[42] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *Proc. of ICLR*, 2016.
[43] 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.
[44] 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.
[45] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. of ICLR*, 2018.
[46] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," 2018, arxiv:1806.01261.
[47] 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.
[48] M. Prates, P. H. C. Avelar, H. Lemos, L. C. Lamb, and M. Y. Vardi, "Learning to solve NP-complete problems: A graph neural network for decision TSP," *Proc. of the AAAI Conf. on AI*, 2019.
[49] 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.
[50] K. Rusek and P. Cholda, "Message-passing neural networks learn Little's law," *IEEE Commun. Lett.*, 2018.
[51] F. Geyer, "DeepComNet: Performance evaluation of network topologies using graph-based deep learning," *Elsevier Performance Evaluation*, vol. 130, 2018.
[52] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio, "RouteNet: Leveraging Graph Neural Networks for network modeling and optimization in SDN," 2019, arxiv:1910.01508.
[53] T. Suzuki, Y. Yasuda, R. Nakamura, and H. Ohsaki, "On estimating communication delays using graph convolutional networks with semi-supervised learning," in *Proc. of IEEE ICOIN*, 2020.
[54] F. Geyer and G. Carle, "The case for a network calculus heuristic: Using insights from data for tighter bounds," in *Proc. of NetCal*, 2018.
[55] D. Starobinski, M. Karpovsky, and L. A. Zakrevski, "Application of network calculus to general topologies using turn-prohibition," *IEEE/ACM Trans. Netw.*, vol. 11, no. 3, 2003.
[56] M. Boyer, "NC-maude: A rewriting tool to play with network calculus," in *Proc. of ISoLA*, 2010.
[57] S. Bondorf and J. B. Schmitt, "Calculating accurate end-to-end delay bounds – you better know your cross-traffic," in *Proc. of EAI ValueTools*, 2015.
[58] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.
[59] P. Erdős and A. Rényi, "On random graphs. i," *Publicationes Mathematicae*, vol. 6, 1959.
[60] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, 2001.
[61] A. Fisher, C. Rudin, and F. Dominici, "Model class reliance: Variable importance measures for any machine learning model class, from the "rashomon" perspective," 2018.
[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, 2011.

ACKNOWLEDGMENTS

Fabien Geyer's work was partially supported by the German-French Academy for the Industry of the Future. Steffen Bondorf's work was partially funded by a Carl-Zeiss Foundation fellowship, carried out in the Distributed Computer Systems Lab at TU Kaiserslautern, Germany, and in the Network Research Laboratory at University of Toronto, Canada. Furthermore, Steffen Bondorf partially carried out this work during the tenure of an ERCIM 'Alain Bensoussan' Fellowship Programme in the Dept. of Information Security and Communication Technology at NTNU Trondheim, Norway, and partially in the Faculty of Mathematics' Center of Computer Science at Ruhr University Bochum, Germany.

Fabien Geyer is currently with Airbus Central Research & Technologies and Technical University of Munich working on methods for network analytics, network performances and architectures. He received the master of engineering in telecommunications from Telecom Bretagne, Brest, France in 2011 and the Ph.D. degree in computer science from Technische Universität München (TUM), Munich, Germany in 2015. His research interests include novel methods for data-driven networking, formal methods for performance evaluation and modeling of networks.

Steffen Bondorf is the Assistant Professor of Distributed and Networked Systems in the Faculty of Mathematics at Ruhr University Bochum, Germany. Steffen received his Dr.-Ing. in Computer Science from TU Kaiserslautern, Germany, in 2016. After graduation, he was a Carl-Zeiss Fellow at TU Kaiserslautern, a research fellow in the School of Computing at National University of Singapore and an ERCIM Fellow in the Dept. of Information Security and Communication Technology at NTNU Trondheim, Norway. Steffen's research interests are in performance modeling and analysis of communication networks.