

# On the Robustness of Deep Learning-predicted Contention Models for Network Calculus

Fabien Geyer

Technical University of Munich | Airbus CRT

Munich, Germany

Steffen Bondorf

Faculty of Mathematics, Center of Computer Science

Ruhr University Bochum, Germany

**Abstract**—The network calculus (NC) analysis takes a simple model consisting of a network of schedulers and data flows crossing them. A number of analysis “building blocks” can then be applied to capture the model without imposing pessimistic assumptions like self-contention on tandems of servers. Yet, adding pessimism cannot always be avoided. To compute the best bound on a single flow’s end-to-end delay thus boils down to finding the least pessimistic contention models for all tandems of schedulers in the network – and an exhaustive search can easily become a very resource intensive task. The literature proposes a promising solution to this dilemma: a heuristic making use of machine learning (ML) predictions inside the NC analysis.

While results of this work were promising in terms of delay bound quality and computational effort, there is little to no insight on when a prediction is made or if the trained algorithm can achieve similarly striking results in networks vastly differing from its training data. In this paper, we address these pending questions. We evaluate the influence of the training data and its features on accuracy, impact and scalability. Additionally, we contribute an extension of the method by predicting the best  $n$  contention model alternatives in order to achieve increased robustness for its application outside the training data. Our numerical evaluation shows that good accuracy can still be achieved on large networks although we restrict the training to networks that are two orders of magnitude smaller.

## I. INTRODUCTION

Deterministic bounds on the end-to-end delay are strictly required in many application areas. Prime examples are data networks in avionics and the automotive industry that are shared between multiple distributed x-by-wire applications [1] as well as safety-critical (factory) systems [2, 3, 4, 5].

Network Calculus (NC) is a versatile framework for the derivation of such bounds. The NC literature provides modeling and analysis tooling such that all steps towards derivation of delay bounds can be taken. There exist results on system modeling, ranging from generic behavior like FIFO [6, 7, 8], non-FIFO [9, 10] or unknown [11] to modern technologies such as IEEE Audio/Video Bridging (AVB) and Time-Sensitive Networking (TSN) [12, 13, 14, 15, 5]. Behavior of such queues, schedulers, shapers etc. are modeled as servers that, in turn, are connected to form a network, the so-called server graph [16, 17].

The server graph carries data flows, exactly one of which will be the designated flow of interest (foi) whose delay

is bounded by the NC analysis. For this network analysis step, results have been created to capture the modeled system behavior as closely as possible. For example, on tandems of work-conserving servers, neither the worst-case burstiness of the foi nor its cross-flows should impact the delay bound computation more than once – i.e., contention for the forwarding resource should not be assumed more pessimistically by the analysis than actually modeled by the server graph. These two core properties of NC are known as pay bursts only once (PBOO) [11, 6, 18, 19] and pay multiplexing only once (PMOO) [20, 21], respectively. Other such refinements try to reduce the amount of mutually exclusive contention assumptions for multiple flows at shared servers (pay segregation only once, PSOO) [22], paying for multiplexing in ring networks less often (pay multiplexing only at convergence points, PMOC) [23], capping the worst-case burstiness with a server’s queue length [24] or using an entirely different alternative to compute bounds on the arrivals of cross-flows [25]. Some of these results are mutually exclusive, e.g., the different arrival bounding method is based on violating the PSOO property.

Correctly capturing these restrictions on realistic worst-case contention in the given model of connected servers does not only help to improve the computed delay bound. It also allows to rank different networks more accurately by not discriminating an alternative that features a design element NC can only consider by pessimistic over-approximation. This allows NC to be used to compare existing network designs to newly proposed ones [2]. However, there is not a single-best NC analysis<sup>1</sup>. In this paper, we focus on networks where there is no knowledge about the multiplexing behavior of flows. This assumption is called arbitrary (or blind) multiplexing. The best delay bound for a flow crossing a cycle-free network of arbitrary multiplexing servers is computed by a specific combination of the properties, the “building blocks”, mentioned above. The exhaustive search for this combination has been improved such that it becomes feasible to execute<sup>2</sup> but it still tends to scale superlinearly with the network size [27]. This search-based analysis is called tandem matching analysis (TMA).

<sup>1</sup>In this paper, we restrict our presentation to the algebraic analysis methods. The optimization analyses in [26, 8] are indeed best w.r.t. to delay bounds but as shown in [27, 8], they tend to become computationally infeasible.

<sup>2</sup>Moreover, its bounds are very close to the optimization approach of [26].

Based on TMA, DeepTMA [28] was recently proposed to alleviate this search-induced problem. DeepTMA is a fast heuristic based on deep learning (DL) that replaces the expensive search with a prediction of the best combination of existing building blocks, i.e., the best contention model. While DeepTMA showed promising results towards fast and accurate NC analysis, understanding how predictions are made remains opaque such that it does not give insights in the NC analysis or the wider applicability of the method. We aim in this paper to address those drawbacks by evaluating the influence of the dataset used in the training phase, as well as its features, on the eventual prediction accuracy. We also contribute an extension of DeepTMA which is able to generate more than one contention model prediction, leading to an increase of the robustness of the method.

We show that DeepTMA is able to cope with scalability, namely that it can be trained on small networks and being used on much larger networks with low impact on the accuracy. Our numerical evaluation illustrates that the relative error of DeepTMA is still below 1% on average when evaluated on networks two orders of magnitude larger than the ones used for training. We also show that training DeepTMA on random networks leads to good applicability on more specific types of networks. Additionally, we give insight into the importance of network features with respect to predicting a contention model. Overall, we first demonstrate DeepTMA’s robustness regarding the relation of training set to evaluated network in terms of size and shape. Finally, we evaluate our extension of DeepTMA that proposes multiple alternative contention models. Our evaluation shows that the robustness of DeepTMA can be increased by generating multiple contention models, leading to a decrease of the error by a factor of 2.

The remainder of the paper is organized as follows: First, we review related work in Section II. Section III introduces Graph Neural Networks (GNNs) and their combination with Network Calculus. In Section IV, we present our extension of DeepTMA and the generation of a dataset to learn from. A numerical evaluation of the robustness of DeepTMA is performed in Section V. Finally, Section VI concludes.

## II. RELATED WORK

Research on combining machine learning with formal methods has been found in a variety of applications, e.g., in theorem proving, model-checking or in SAT-SMT problems. In the following, we aim to provide a focused depiction of efforts that are interesting and related to our work. Namely, the performance in networks and Graph Neural Networks (GNNs). A more comprehensive survey on machine learning-assisted formal methods can be found in [29].

GNNs were first introduced in [30, 31], a concept subsequently refined in recent works. Message-passing neural network were introduced in [32], with the goal of unifying various GNN and graph convolutional concepts. [33] formalized graph attention networks, which enables to learn edge weights of a node neighborhood. Finally, [34] 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 [35, 32], solving the traveling salesman problem [36], prediction of satisfiability of SAT problems [37], or basic logical reasoning tasks and program verification [38]. For computer networks, they have recently been applied to prediction of average queuing delay [39], different non-NC-based performance evaluations of networks [40, 41, 42], and routing [43]. In the realm of NC, there is surprisingly little work as of yet. Predating DeepTMA [28] 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 [44]. This was developed into DeepTMA that can provide multiple predictions per analysis. Independent efforts aim at predicting delay bounds, too. This work [45, 46] uses supervised learning and benchmarks the predictions against a NC-based analysis. Another similar goal to our work is to provide small yet controllable computation times to make the proposed analysis fit for application in design space exploration.

Regarding assessing the robustness of GNNs, [37, 36] showed that GNNs can be trained on a given set of graphs while being able to extrapolate on other types or much larger graphs. Finally, [47] recently proposed an approach to explain predictions from GNNs, by reducing the input graphs to subgraphs containing a small subset of nodes which are most influential for the prediction.

## III. BACKGROUND: GRAPH NEURAL NETWORK FOR NC

We give a brief overview of the DeepTMA heuristic in this section. We refer the reader to [28] for the full formulation of the method. It is based on the concept of Graph Neural Network (GNN) introduced in [30, 31]. 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 1. Since the delay bounds are still computed using the formal network calculus analysis, they inherit its proven correctness.

### 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 [30, 31].

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

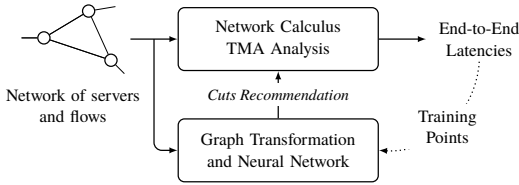


Figure 1: Overview of the proposed approach.

behind GNNs is called *message passing*, where hidden states of nodes  $\mathbf{h}_v$  (i.e. vectors of real numbers) are iteratively passed between neighboring nodes.

At each iteration of message passing, each node in the graph aggregates the hidden states of its neighbors, use this aggregate to update its own hidden state, and sends the updated state at the next iteration:

$$\mathbf{h}_v^{(t=0)} = \text{init}(\mathbf{i}_v) \quad (1)$$

$$\mathbf{h}_v^{(t+1)} = \text{aggr}\left(\left\{\mathbf{h}_u^{(t)} \mid u \in \text{NBR}(v)\right\}\right) \quad (2)$$

with  $\mathbf{h}_v^{(t)}$  representing the hidden state of node  $v$  at iteration  $t$ , *aggr* a function which aggregates the set of hidden states of the neighboring nodes  $\text{NBR}(v)$  of  $v$ , and *init* a function for initializing the hidden states based on the input features. Those hidden states are propagated throughout the graph using multiple iterations of Equation (2) until a fixed point is found. The final hidden state is then used for predicting properties about nodes:

$$\mathbf{o}_v = \text{out}\left(\mathbf{h}_v^{(t \rightarrow \infty)}\right) \quad (3)$$

with *out* a function transforming  $\mathbf{h}$  to the target values.

In GNNs, the aggregation of hidden states corresponds to their sum, and the *aggr* and *out* functions are feed-forward neural networks (FFNN) such that:

$$\mathbf{h}_v^{(t+1)} = \text{aggr}\left(\left\{\mathbf{h}_u^{(t)} \mid u \in \text{NBR}(v)\right\}\right) = \text{aggr}\left(\sum_{u \in \text{NBR}(v)} \mathbf{h}_u^{(t)}\right) \quad (4)$$

Various extensions of GNNs have been recently proposed in the literature. We selected Gated Graph Neural Networks (GGNN) [38] for implementing DeepTMA, extended with an attention mechanism similar to the one proposed in [33]. This extension implements *aggr* using a recurrent unit and unrolls Equation (2) 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.

In order to propagate the hidden states throughout the complete graph, a fixed number of iterations is done. This extension has been shown to outperform the original GNNs that required to run the iteration until a fixed point was found. We refer to [34] for additional details on GNNs.

### B. Application to TMA

In order to apply the concepts described in Section III-A to a network calculus analysis, we transform a simple NC network (server graph with one data flow) into a graph. Figure 2 illustrates this transformation.

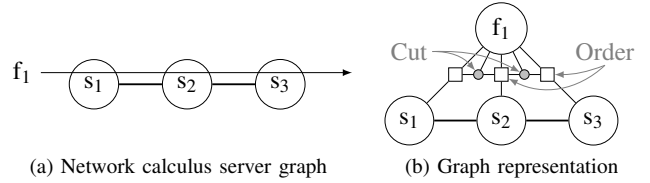


Figure 2: Graph representation of a sample tandem network.

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. The path taken by a flow in this graph is encoded using undirected edges that 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 has an impact on dependency structures and contention models. TMA “cuts” the network between pairs of servers. The potential TMA cuts between on the path traversed by the flow of interest are represented nodes. A cut node is connected via edges to its flow and to its pair of servers.

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 need to comprise some NC definitions. A comprehensive treatment of NC can be found in [11]. TMA and DeepTMA are described in [27] and [28], respectively. NC resource models rely on non-negative, wide-sense increasing functions

$$\mathcal{F}_0 = \{f : \mathbb{R} \rightarrow \mathbb{R}_\infty^+ \mid f(0) = 0, \forall s \leq t : f(s) \leq f(t)\},$$

where  $\mathbb{R}_\infty^+ := [0, +\infty) \cup \{+\infty\}$ . These functions pass through the origin. The functions  $A(t)$  and  $A'(t)$  cumulatively count input and output data (in absolute time  $t$ ) and are the basis of NC curves (interval time  $d$ ). The server graph crossed by flows, in short network, is annotated with the following curves, each one bounding a relevant property.

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

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

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 scheduler, queue, etc. 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  $\beta$ .

Curves are in  $\mathcal{F}_0$ , too, and will thus not permit for data occurrences or resource availability in time intervals of duration 0. They are incorporated as input features as follows:

- For each server  $s$ , parameters of its rate-latency service curve are used where  $\beta_s(d) = \max\{0, \text{rate}_s \cdot d - \text{latency}_s\} : [\text{rate}_s, \text{latency}_s]$ .
- For each flow  $f$ , parameters of its token bucket arrival curve are used where  $\alpha_f(d) =$

$\{rate_f \cdot d + burst_f\}_{\{d>0\}}$  (i.e.,  $\alpha_f(d) = 0$  for  $d \leq 0$ ):  $[rate_f, burst_f]$ .

- For each path ordering node  $p$ , the hop count is encoded as an integer  $PathOrder$ .
- Finally, cut nodes do not have input features.

Note, that in case more complex arrival or service curve types than affine curves [48] 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 DeepTMA is to predict which tandem decomposition will result in the tightest bound, only the nodes presenting cuts have output features. This problem 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 don't:  $[cut, \overline{cut}]$ . The overall prediction to be fed back, i.e., the selection of one out of TMA's potential decompositions for a given fo's path, is defined by the set of all  $cut$  classifications for this path.

#### IV. INCREASING THE ROBUSTNESS OF DEEPTMA

##### A. $DeepTMA_n$ : Generating multiple tandem decompositions

Given a fo and a potential cut location, the output of the neural network is a probability of cutting. This probability is generated by the neural network using the sigmoid function after its last layer. A set of cuts is also called a "tandem decomposition". In case a single tandem decomposition has to be generated, the decision of cutting is made using a threshold of 50%.

Those cut probabilities may also be used to generate multiple tandem decompositions as illustrated in Algorithm 1. In case the number of all potential tandem decompositions is lower than the number of requested ones, we simply return all of them. Otherwise, we sample the distribution of cuts in order to generate the decompositions. We label this extension of DeepTMA as  $DeepTMA_n$ , with  $n$  the number of tandem decompositions generated.

---

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

---

```

if  $n \leq L^2$  then return all combinations of cuts
else
  for all  $i := 1$  to  $n$  do
     $v \leftarrow [c_1, \dots, c_L] \sim \mathcal{U}(0, 1)^L$ 
     $cuts_i \leftarrow \mathbb{I}(v \leq [\Pr(cut_{foi,1}^{GNN}), \dots, \Pr(cut_{foi,L}^{GNN})])$ 
    ( $\mathbb{I}$  is the indicator function)
  return  $\{cuts_1, \dots, cuts_n\}$ 

```

---

##### B. Dataset generation

In order to train our neural network architecture, we randomly generated a set of topologies according to three different random topology generators: *a*) tandems or daisy-chains, *b*) trees and *c*) random server graphs following the  $G(n, p)$  Erdős–Rényi model [49]. For each created server, a rate latency service curve was generated with uniformly random rate and latency parameters. A random number of flows is

generated with random source and sink servers. 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, 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 V. Table I summarizes different statistics about the generated dataset. The dataset is available online to reproduce our learning results<sup>3</sup>. Note that compared to the original dataset used for training DeepTMA [28], this dataset contains larger networks.

| 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 I: Statistics about the randomly generated dataset.

Additionally to this dataset, we also evaluate our approach on the set of networks used in [27]. Table II summarizes different statistics about the generated dataset. Compared to the dataset used for training, this additional set of networks is up to two orders of magnitude larger in terms of number of servers and flows per network. This property will be used in Section V to investigate if our approach is able to scale to such larger networks, both in terms of accuracy and execution time, without tailoring the training set.

| 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 II: Statistics about the set of networks from [27].

#### V. NUMERICAL EVALUATION

We evaluate in this section our extensions of DeepTMA as well as its robustness and scalability. Via a numerical evaluation, we illustrate the tightness and execution time of DeepTMA and highlight its usability for practical use-cases.

In order to numerically evaluate and compare DeepTMA against TMA, we selected the relative error metric as our main metric for the rest of this evaluation. This metric measures the relative difference of DeepTMA against TMA with respect to the end-to-end delay bound, and is defined for flow  $f_i$  as:

$$RelErr_{f_i} = \frac{Delay_{f_i}^{DeepTMA} - Delay_{f_i}^{TMA}}{Delay_{f_i}^{TMA}} \quad (5)$$

<sup>3</sup><https://github.com/fabgeyer/dataset-deeptma-extension>

### A. Impact of training network sizes on error

We first investigate the scalability of DeepTMA by evaluating the impact of the training dataset on the accuracy of the method. We trained here two additional instances of the deep-learning part of DeepTMA. Each has a different restriction on the maximum amount of flows in the networks to be included in the training set, namely 50 and 100.

Figure 3 illustrates the error of those additional instances of DeepTMA compared to the one trained on the full dataset. There is an evident trend that a smaller training set size as imposed by our restriction generally leads to an increasing relative error. However, there is one exception at path length 17. This illustrates that the “quality” of the training set can be more important than its size. We provide a closer look at network type and features as potential impact factors for the training set quality in the Sections V-B and V-C.

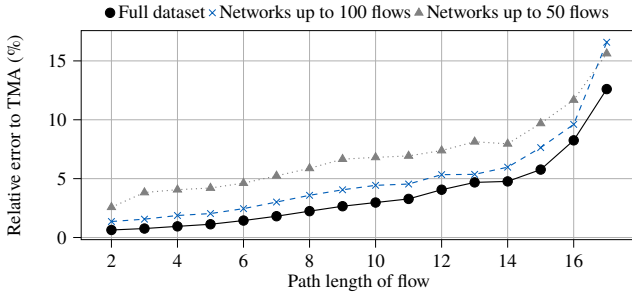


Figure 3: Impact of training size on relative error of DeepTMA.

Similarly, Figure 4 illustrates the impact of training dataset on the set of networks from [27]. The difference between the different variants of DeepTMA is minimal except on the large networks. This indicates that DeepTMA is still able to scale, even when trained on much smaller networks. Moreover, we can see small datasets outperforming the full set again in some cases.

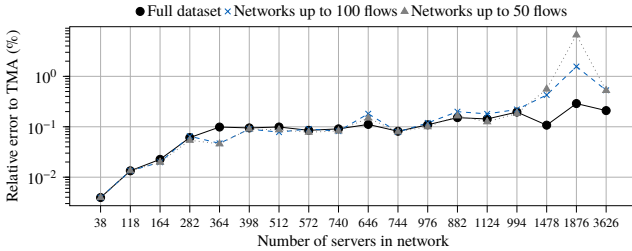


Figure 4: Impact of training size on the set of networks from [27].

### B. Influence of network type used for training

We examine in this section the impact of the network types used for training DeepTMA on its accuracy. As explained in Section IV-B, three different types of networks were generated, namely *a)* tandems, *b)* trees and *c)* random server graphs based on the  $G(n, p)$  Erdős–Rényi model.

We evaluate the ability of DeepTMA to extrapolate on other networks by training three different variants for DeepTMA, each on one type of networks. Results are presented in Figure 5. Compared DeepTMA trained on the full dataset, training only on tandem or tree networks leads to good ability at extrapolating on other types of networks. Surprisingly, tree network-based training dataset is outperformed by the tandem-based one that, in turn, is very competitive with the random server graphs.

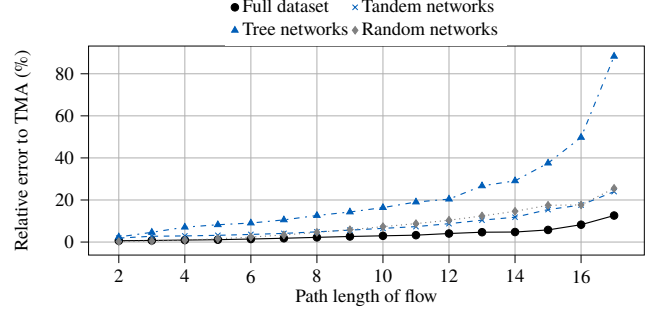


Figure 5: Impact of network types used for training on relative error of DeepTMA.

### C. Importance of features and locality

In order to better understand the importance of the input features used in DeepTMA, we assess each feature’s importance following the permutation-based importance measure [50, 51]. For each input feature presented in Section III-B, we randomize it by randomly permuting its values in the training set, and assess the impact it has on the accuracy of the predictions. We define the importance metric as:

$$Importance(Feature) = \frac{1}{|\mathcal{F}|} \sum_{f_i \in \mathcal{F}} \left( RelErr_{f_i}^{Feature} - RelErr_{f_i}^{Baseline} \right) \quad (6)$$

with the baseline corresponding to DeepTMA 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.

Features importance are presented in Figure 6(above). The service rate of the servers in the network have the largest influence on the final decision of cutting. Such behavior confirms an existing result of NC, which is known to be sensitive to service rates. The remaining features appear to have less importance on the cut prediction. 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 [52]) 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 (2)) and the impact it has on the relative error. As for feature importance, we compare the results according to Equation (6). Results are

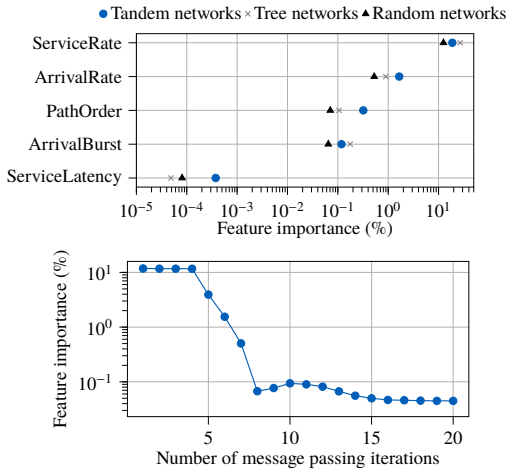


Figure 6: (above) Feature importance for DeepTMA and (below) impact of number of GNN message passing iterations.

presented in Figure 6(below). The first 4 loop iterations appear to have the largest influence 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.

#### D. Evaluation of DeepTMA<sub>n</sub>

We now start focusing on actively improving robustness and evaluate our extension of DeepTMA defined in Section IV-A. It enables DeepTMA to generate more than one tandem decomposition. Results are presented in Figure 7(a), where the subscript  $n$  denotes the number of tandem decompositions generated by DeepTMA <sub>$n$</sub> . To benchmark DeepTMA <sub>$n$</sub> , we depict the performance of a random heuristic in Figure 7(b).

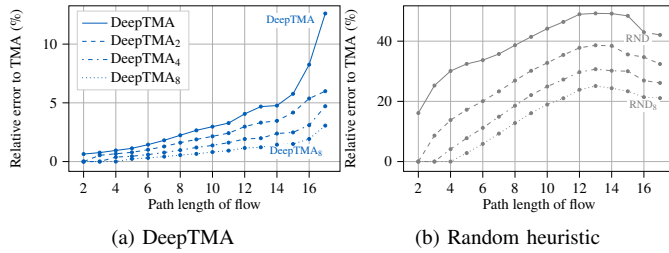


Figure 7: Evaluation of DeepTMA<sub>n</sub> and random heuristic.

As expected, the generation of more than one tandem decomposition results in a decrease of the error. Already with DeepTMA<sub>2</sub>, the error is reduced by a factor of 2 on the larger networks. This illustrates that the robustness of DeepTMA can be increased by using Algorithm 1, even at the smallest additional computational cost of going to DeepTMA<sub>2</sub>. The

random heuristic performs considerably worse in all aspects evaluated above.

#### E. Scalability on large networks

We evaluate in this section the robustness of DeepTMA and DeepTMA<sub>n</sub> with respect to scalability. The networks from [27] are evaluated here, since those networks are almost two orders of magnitude larger than the networks used for training the DeepTMA GNN, as shown in Tables I and II.

Figure 8 illustrates the relative error of DeepTMA and DeepTMA<sub>n</sub> compared to a random heuristic which selects the tandem decompositions randomly. The family of DeepTMAs achieve relative errors that are two orders of magnitudes smaller than the random heuristics, resulting in better end-to-end delay bound tightness w.r.t. the exhaustive TMA.

Although DeepTMA wasn’t trained on such large networks, the relative error still stays below 0.3% even on the larger networks. DeepTMA<sub>8</sub> is even able to reach relative errors below 0.02%, indicating a good ability to scale.

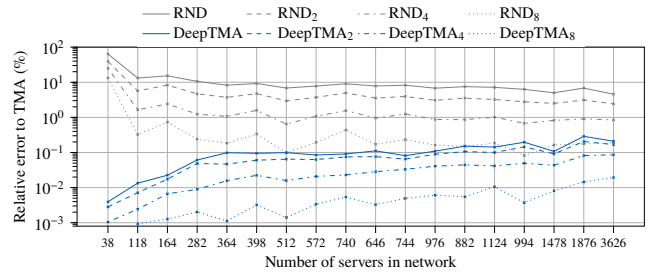


Figure 8: Evaluation of DeepTMA<sub>n</sub> on the set of networks from [27].

## VI. CONCLUSION

We contributed in this paper an extension of DeepTMA for generating multiple tandem decomposition predictions and a comprehensive assessment of its robustness in terms of scalability and impact of training data on its accuracy. We also provided some insights on which feature is important for making a prediction.

Via a numerical evaluation we showed that DeepTMA can be trained on small networks and still provide good accuracy on much larger networks, up to two order of magnitude larger in terms of number of servers and flows. We also showed that the network type used for training can have a large impact on the accuracy of the prediction made by the GNN. Nevertheless, we showed that training DeepTMA on randomly generated networks can still lead to good accuracy, suggesting that tailoring the training data to more realistic use-cases might not be necessary for application on real networks.

Those new results indicate that DeepTMA is able to generalize tandem decomposition rules from small random networks which can also be applied to larger networks, at a low execution time cost. Finally, we also proposed an extension of DeepTMA which is able to generate multiple predictions, decreasing the prediction error by a factor of two.

## 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] A. Amari, A. Mifdaoui, F. Frances, and J. Lacan, "Worst-case timing analysis of AeroRing – a full duplex ethernet ring for safety-critical avionics," in *Proc. of IEEE WFCs*, 2016.
- [3] A. Finzi, A. Mifdaoui, F. Frances, and E. Lochin, "Incorporating TSN/BLS in AFDX for mixed-criticality applications: Model and timing analysis," in *Proc. of IEEE WFCs*, 2018.
- [4] A. Finzi and S. S. Craciunas, "Integration of SMT-based scheduling with rc network calculus analysis in TTEthernet networks," in *Proc. of IEEE ETFA*, 2019.
- [5] J. Zhang, L. Chen, T. Wang, and X. Wang, "Analysis of TSN for industrial automation based on network calculus," in *Proc. of IEEE ETFA*, 2019.
- [6] M. Fidler, "Extending the network calculus pay bursts only once principle to aggregate scheduling," in *Proc. of QoS-IP*, 2003.
- [7] L. Bisti, L. Lenzi, E. Mingozzi, and G. Stea, "Numerical analysis of worst-case end-to-end delay bounds in fifo tandem networks," *Real-Time Syst.*, 2012.
- [8] 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.
- [9] G. Rizzo and J.-Y. Le Boudec, "'pay bursts only once' does not hold for non-FIFO guaranteed rate nodes," *Perform. Eval.*, vol. 62, no. 1-4, 2005.
- [10] 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.
- [11] J.-Y. Le Boudec and P. Thiran, *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, 2001.
- [12] R. Queck, "Analysis of ethernet avb for automotive networks using network calculus," in *Proc. of IEEE ICVES*, 2012.
- [13] J. A. R. De Azua and M. Boyer, "Complete modelling of avb in network calculus framework," in *Proc. of RTNS*, 2014.
- [14] 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.
- [15] H. Daigmorte, M. Boyer, and L. Zhao, "Modelling in network calculus a TSN architecture mixing time-triggered, credit based shaper and best-effort queues," 2018, working paper or preprint.
- [16] S. Bondorf and J. B. Schmitt, "The DiscoDNC v2 – a comprehensive tool for deterministic network calculus," in *Proc. of EAI ValueTools*, 2014.
- [17] B. Cattelan and S. Bondorf, "Iterative design space exploration for networks requiring performance guarantees," in *Proc. of IEEE/AIAA DASC*, 2017.
- [18] G. Chen, K. Huang, C. Buckl, and A. Knoll, "Applying pay-burst-only-once principle for periodic power management in hard real-time pipelined multiprocessor systems," *ACM Trans. Des. Autom. Electron. Syst.*, 2015.
- [19] Y. Tang, Y. Jiang, X. Jiang, and N. Guan, "Pay-burst-only-once in real-time calculus," in *Proc. of IEEE RTCSA*, 2019.
- [20] 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.
- [21] A. Bouillard, B. Gaujal, S. Lagrange, and É. Thierry, "Optimal routing for end-to-end guarantees using network calculus," *Performance Evaluation*, 2015.
- [22] S. Bondorf and J. B. Schmitt, "Should network calculus relocate? an assessment of current algebraic and optimization-based analyses," in *Proc. of QEST*, 2016.
- [23] A. Amari and A. Mifdaoui, "Worst-case timing analysis of ring networks with cyclic dependencies using network calculus," in *Proc. of IEEE RTCSA*, 2016.
- [24] S. Bondorf and J. B. Schmitt, "Improving cross-traffic bounds in feed-forward networks – there is a job for everyone," in *Proc. of GI/ITG MMB & DFT*, 2016.
- [25] S. Bondorf, P. Nikolaus, and J. B. Schmitt, "Catching corner cases in network calculus – flow segregation can improve accuracy," in *Proc. of GI/ITG MMB*, 2018.
- [26] 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.
- [27] 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.
- [28] F. Geyer and S. Bondorf, "DeepTMA: Predicting effective contention models for network calculus using graph neural networks," in *Proc. of INFOCOM*, 2019.
- [29] M. Amrani, L. Lúcio, and A. Bibal, "ML + FV =  $\heartsuit$ ? A survey on the application of machine learning to formal verification," 2018, arxiv:1806.03600.
- [30] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proc. of IEEE IJCNN*, 2005.
- [31] 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.
- [32] 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.
- [33] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. of ICLR*, 2018.
- [34] 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.
- [35] 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.
- [36] 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 Conference on Artificial Intelligence*, vol. 33, pp. 4731–4738, 2019.
- [37] 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.
- [38] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *Proc. of ICLR*, 2016.
- [39] K. Rusek and P. Cholda, "Message-passing neural networks learn little's law," *IEEE Communications Letters*, 2018.
- [40] F. Geyer, "Performance evaluation of network topologies using graph-based deep learning," in *Proc. of EAI ValueTools*, 2017.
- [41] —, "DeepComNet: Performance evaluation of network topologies using graph-suited deep learning," *Performance Evaluation*, 2018.
- [42] 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.
- [43] F. Geyer and G. Carle, "Learning and generating distributed routing protocols using graph-based deep learning," in *Proc. Big-DAMA*, 2018.
- [44] —, "The case for a network calculus heuristic: Using insights from data for tighter bounds," in *Proc. of NetCal*, 2018.
- [45] T. L. Mai, N. Navet, and J. Migge, "A hybrid machine learning and schedulability analysis method for the verification of TSN networks," in *Proc. of IEEE WFCs*, 2019.
- [46] —, "On the use of supervised machine learning for assessing schedulability: Application to ethernet TSN," in *Proc. of RTNS*, 2019.
- [47] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," in *Proc. of NeurIPS*, 2019.
- [48] A. Bouillard and É. Thierry, "An algorithmic toolbox for network calculus," *Discrete Event Dynamic Systems*, vol. 18, no. 1, 2008.
- [49] P. Erdős and A. Rényi, "On random graphs. i," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.
- [50] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001.
- [51] A. Fisher, C. Rudin, and F. Dominici, "Model class reliance: Variable importance measures for any machine learning model class, from the "rashomon" perspective," 2018.
- [52] 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.