2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS) | 978-1-6654-0386-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/RTAS52030.2021.00021

Tightening Network Calculus Delay Bounds by Predicting Flow Prolongations in the FIFO Analysis

Fabien Geyer*† Alexander Scheffler‡ Steffen Bondorf‡

*Technical University of Munich
Munich, Germany

†Airbus Central R&T
Munich, Germany

‡Faculty of Mathematics, Center of Computer Science
Ruhr University Bochum, Germany

***Abstract*—Network calculus offers the means to compute worst-case traversal times based on interpreting a system as a queueing network. A major strength of network calculus is its strict separation of modeling and analysis frameworks. That is, a model is purely descriptive and can be put into multiple different analyses to derive a data flow's worst-case traversal time bound. One of the recent results in this category is the so-called flow prolongation. Flow prolongation actively manipulates the internal model of the analysis by virtually extending the path of flows, i.e., by deliberately creating a more pessimistic setting of resource contention between flows. It was shown that flow prolongation can theoretically decrease worst-case traversal time bounds under certain assumptions. Yet, due to its exhaustive search, it was also shown that flow prolongation does not scale and it might not even have an impact in larger queueing networks. In this paper we introduce DeepFP, an approach to make the analysis scale by predicting flow prolongations using a graph neural network. In our evaluation, we show that DeepFP can improve results in networks of FIFO queues considerably, where the delay bound can be reduced by 13.7 % in large FIFO networks at negligible additional cost on the execution time of the analysis.**

I. Introduction

Nowadays, many newly developed networked systems aim to provide some kind of performance guarantee – prime examples are those in the automotive and avionics sector [1] as well as factory automation [2]. Applications in these domains that rely on network performance care about one important property: the worst-case traversal time, i.e., the end-to-end delay, of data communication. Safety-critical applications that are crucial for the entire system's certification even need to show guaranteed upper bounds on the end-to-end delay.

Network Calculus (NC) offers a framework for this purpose. It consists of two parts: modeling and analysis. For best results, i.e., tight delay bounds, both should be developed in lockstep to prevent mismatches in their capabilities. Unfortunately, this has not always been the case and the analysis needs to catch up. Some easy to model network characteristics such as multicast flows [3] or ring topologies [4, 5] have only seen more detailed treatment recently. Thanks to the analysis' independence of the descriptive model, other characteristics also found their way into the analysis. Most prominent are the properties Pay Bursts Only Once (PBOO) [6] and Pay Multiplexing Only Once (PMOO) [7] that prevent the analysis from assuming data flows to exhibit stop-and-go behavior and/or overtake each other multiple times when crossing a sequence of servers (so-called tandems). In general, improvements to the NC tandem analysis tried to remove pessimistic assumptions

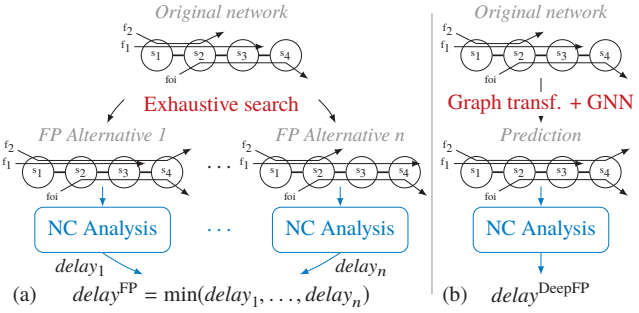

Figure 1: Comparison between the (a) original FP [11] with $O(n^m)$ NC analyses and (b) our DeepFP with one prediction.

from its internal model in order to improve the derived delay bounds for the original, user-provided model. Unfortunately, this also lead to the situation that none of the fast, algebraic NC analyses is strictly best. A search for the most suitable analysis is often advised [8, 9, 10]. Novel analysis features try to narrow down the amount of potentially best analyses.

An entirely different approach was recently presented with the Flow Prolongation (FP) feature [11]. It actively converts the network model given to the NC analysis to a more pessimistic one that circumvents limitations of the NC analysis capabilities. The analysis derives algebraic NC terms bounding a flow's delay. The amount of terms grows exponentially with the network size and none of them computes the tightest bound for all data flow descriptions. All need to be derived and solved [9]. The analysis has to compute a multitude of valid delay bounds to find the minimum among them. FP not only increases the amount of algebraic terms (and thus bounds), it also complicates the prediction of a term's added pessimism.

FP is conceptually straight-forward: assume cross-flows take more hops than they actually do. Nonetheless FP is a powerful feature to add to a NC analysis, it was even adopted in the Stochastic Network Calculus [12]. Unfortunately, finding the best prolongation alternative is prone to a combinatorial explosion. On each tandem of length n with m cross-flows, there are $O(n^m)$ alternatives to prolong flows. Even with a deep understanding of the NC analysis applied to reduce FP alternatives it could not be made to scale to larger models [11].

A novel approach to overcome exhaustive searches in the algebraic NC analysis was recently proposed: Graph Neural Network (GNN) predictions for NC term creation. This can

978-1-6654-0386-3/21/$31.00 ©2021 IEEE
DOI 10.1109/RTAS52030.2021.00021

be used to make the NC analysis scale by restricting the exhaustive search to few alternatives [13, 14]. We base our contribution on this work, presenting *DeepFP* illustrated in Figure 1.

By demonstrating that we can make the FP analysis scale this way, we also reveal that its impact on the derived delay bound is very sensitive to the network model's assumptions. The foremost contribution of this paper is the FP analysis of FIFO networks. Under this assumption and applying the state-of-the-art algebraic Least Upper Delay Bound (LUDB) analysis [15, 16], we derive entirely new conditions for beneficial flow prolongations, train the GNN and acquire significantly improved delay bounds.

Our results can be applied to any system designed around FIFO-multiplexing and -forwarding of data. Most notable are Ethernet-based networks like Avionics Full-DupleX Ethernet (AFDX) or IEEE Time-Sentitive Networking (TSN). Even though they follow the "FIFO per priority queue" design, their NC model is essentially a FIFO system model. Our results can be combined with existing works on service modeling of the specific schedulers used in those systems.

This paper is organized as follows: Section II presents the related work and Section III gives an overview on NC analyses. In Section IV, we show how FP can improve bounds in FIFO multiplexing networks as well as the challenge it imposes. Section V provides the DeepFP method to make FP applicable to a wide range of networks. Section VI evaluates DeepFP and Section VII concludes the paper.

II. Related work

NC and RTC: Network Calculus takes a purely descriptive model of a network of queueing locations and data flows (see Appendix A). The NC analysis then computes a bound on the worst-case delay for a certain flow, the flow of interest (foi) (see Appendix B). A variant of NC that focuses on (embedded) real-time systems is the so-called Real-Time Calculus (RTC) [17]. Equivalence between the slightly differing resource descriptions has been proven in [18]. What remains is the difference in modeling of the "network" and the analysis thereof. RTC models networks of components such as the Greedy Processing Component (GPC) [19, 20]. Each component represents a macro, i.e., a fixed sequence of algebraic NC operations to apply to its input. Thus, the model already encodes the analysis. Moreover, this component modeling mostly restricts the analysis to strict priority multiplexing, yet, efforts to incorporate advanced properties such as PBOO and PMOO can be found in the literature [21, 22]. We, in contrast, aim for a model-independent improvement of the automatic derivation of a valid order of NC operations – the process called NC analysis – for networks of FIFO multiplexing systems. First results on this topic in NC [6, 23] were refined to the LUDB analysis [15, 16]. Later works entirely replace the algebraic NC analysis with an optimization one [24, 25], a mixed integer linear programming formulation that introduces forbiddingly large computational effort. Current efforts try to improve it by trading off delay bound tightness [26].

Flow Prolongation (FP): We pair FP with LUDB's DEBORAH tool to counteract its main tightness-compromising problem, thus considerably improving delay bounds. There has been one previous mention of FP in FIFO networks: [15] briefly shares the observation that, if prolonged, a cross-flow can be aggregated with the foi – independent of the LUDB problem we tackle. This can be combined with our contribution. We leave its investigation to future work.

Prolonging at the front may also be possible, but only in the arbitrary multiplexing PMOO analysis [27].

Graph Neural Networks: GNNs were first introduced in [28, 29] and [30] presents a framework that formalizes many concepts applied in GNNs in a unified way. GNNs were already proposed as an efficient method for speeding up exhaustive searches or similar NP-hard problems such as the traveling salesman problem [31]. A recent survey [32] 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 (SAT) or satisfiability modulo theories (SMT) problems.

For computer networks, they have recently been applied to prediction of average queuing delay [33] and different non-NC performance evaluations of networks [34, 35, 36, 37]. [38] recently used GNNs for predicting the feasibility of scheduling configurations in Ethernet networks.

NC and GNN: DeepTMA was proposed in [13, 39] as a framework where GNNs were used for predicting the best contention model to use whenever there are alternatives for a tandem. DeepFP and DeepTMA are closely related: both methods use a graph transformation and a GNN to replace a computationally expensive exhaustive search. While DeepTMA targeted the Tandem Matching Analysis (TMA) [9], DeepFP focuses on FP and therefore we need to design a different graph transformation to connect NC and the GNN. Moreover, DeepTMA was shown to scale to large networks with up to 14 000 flows [39] in follow-up improvements of the method.

III. Network Calculus Analyses

The main objective of NC is to derive a bound on the flow of interst's (foi's) end-to-end delay, subject to interference and queuing. The resulting order of data in a shared queue when two different flows multiplex is a main concern of the NC analysis. NC generally differentiates between no assumption at all, so-called arbitrary multiplexing, and FIFO multiplexing. Given curves β lower bounding available forwarding service and α upper bounding arriving data (see Appendix A), NC can compute lower bounds on a foi's residual service.

Theorem 1 (Residual Service Curve): Consider a server s that offers a strict service curve β. Assume flows f_1 and f_2 with arrival curves α_1 and α_2, respectively, traverse the server. We can compute the service curve for guaranteed residual service for f_1, subject to multiplexing of flows at s, as

$$\beta^1(t) = [\beta(t) - \alpha_2(t)]^{\uparrow} =: \beta \ominus \alpha_2 \tag{1}$$

for arbitrary multiplexing and as

$$\beta^1(t,\theta) = [\beta(t) - \alpha_2(t-\theta)]^{\uparrow} \cdot \mathbf{1}_{\{t>\theta\}} =: \beta \ominus_\theta \alpha_2, \forall \theta \geq 0 \tag{2}$$

for FIFO multiplexing [40, Theorem 4]. $\mathbf{1}_{\{\text{condition}\}}$ denotes the indicator function (1 if the condition is true, 0 otherwise) and $[g(x)]^{\uparrow} = \sup_{0 \le z \le x} g(z)$ is the non-decreasing closure of function $g(x)$ defined on positive real values.

In a FIFO multiplexing server, the residual service depends on the flow of interest (f_1 in Theorem 1). Yet, it is desired to be computed seemingly independent of it as with arbitrary multiplexing. θ encodes the FIFO worst cases for any foi. It thus defines an infinite set of (valid) residual service curves.

A. PMOO, the Analysis for Arbitrary Multiplexing

An important discovery in the evolution of analysis capabilities was that, even assuming arbitrary multiplexing, cross-flows' worst-case burstiness need not be assumed to fully collide with the foi at each server. This is known as Pay Multiplexing Only Once (PMOO) [7]. To achieve this, the proposed PMOO analysis computes a residual service curve for an entire tandem of servers.

The PMOO analysis has a disadvantage when analyzing feed-forward networks. To handle demultiplexing on the foi's path, cross-flows interfering on different foi-subpaths need to be analyzed in a demultiplexed fashion in the entire feed-forward analysis. At shared servers before the foi's path, that creates mutually exclusive worst-case assumptions [41]. For an example, see Figure 2 where at server s_1 each cross-flow, f_1 and f_2, would compute a residual service curve to demultiplex from the other.

B. DEBORAH, the Analysis Tool for FIFO Multiplexing

For the analysis of FIFO multiplexing tandems, there are two challenges:
a) implementing the PMOO principle and
b) finding the best setting for the free θ parameter in the residual service curve computation.

The Least Upper Delay Bound (LUDB) analysis [42, 15, 16] tackles both. As its name suggests, *b)* is achieved by finding the smallest among many alternative delay bounds (similar to Figure 1(a)). To do so, LUDB converts the problem of setting all θ in the algebraic NC term into several linear programs. This conversion strictly requires the modeling curves to be affine, a restriction we inherit in this paper.

The more important part for our flow prolongation is the current solution to challenge *a)*. LUDB does not necessarily achieve a full implementation of the PMOO principle. It is very susceptible to the nesting of flows on the analyzed tandem. In general, a tandem is called nested if any two flows have disjunct paths or one flow is completely included in the path of the other flow. For example, in Figure 2(a), f_2 is completely included in the path of f_1 but neither are completely included in the foi's path. If flows form a non-nested interference pattern as f_1 and f_2 in Figure 3(a), then LUDB needs to cut the tandem into a sequence of sub-tandems, each with a nested interference pattern. At these cuts, a tightness-reducing computation to bound the arrivals of cross-flows needs to be executed. It adds the cross-flow burstiness to all sub-tandem residual service curve computations and

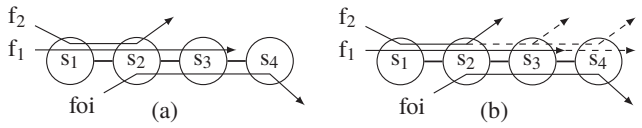

Figure 2: (a) Example tandem network shown in Figure 1 and (b) indication of all its potential flow prolongation alternatives

PMOO is not achieved. Therefore, we aim at reducing the amount of cuts.

Last, the DEBORAH tool has been developed to implement LUDB [43] for tandem networks only. We extended it to analyze feed-forward networks in our numerical evaluations.

C. Flow Prolongation

Flow Prolongation was designed as an add-on feature to mitigate the PMOO analysis's problem described above [11]. FP is, however, a generic approach that is independent of any multiplexing assumption. More formally, it is defined by:

Corollary 1 (Delay Increase due to FP): Assume a tandem $\mathcal{T}$ defined by the foi's path. Let the foi be f_1 and let there be cross-flows on $\mathcal{T}$. A prolongation of cross-flows to create tandem $\mathcal{T}_{\text{FP}}$ increases the end-to-end delay of f_1 on $\mathcal{T}_{\text{FP}}$.

Proof 1: Wlog assume a single cross-flow f_2 to be prolonged over one additional server s where f_1 is present, too. Compared to $\mathcal{T}$, s in $\mathcal{T}_{\text{FP}}$ multiplexes incoming data of f_2 with data of f_1 in its queue. s either forwards this data of f_2 after f_1, causing no increase of f_1's delay on $\mathcal{T}_{\text{FP}}$, or it forwards at least parts of the data of f_2 before f_1, causing an additional queuing delay to f_1.

Corollary 1 shows that FP is a conservative transformation adding pessimism to the network model that increases the foi's delay. For delay bounds, it holds that:

Corollary 2 (FP Delay Bound Validity): Assume a tandem $\mathcal{T}$ defined by the flow of interest's path. Let $\mathcal{T}_{\text{FP}}$ be derived from $\mathcal{T}$ by flow prolongation. Then, the bound on the foi's worst-case delay in $\mathcal{T}_{\text{FP}}$ is a bound on the foi's delay on $\mathcal{T}$.

Proof 2: Per Corollary 1, we know that the foi's end-to-end delay will not decrease by FP. Thus, the tight delay bound in $\mathcal{T}_{\text{FP}}$ will exceed the tight delay bound on $\mathcal{T}$ and any potentially untight bound derived for $\mathcal{T}_{\text{FP}}$ bounds the foi's delay on $\mathcal{T}$.

Take the sample tandem in Figure 2(a), where bounding the arrivals of data flows f_1 and f_2 is required at their first location of interference with the foi, server s_2. Assuming arbitrary multiplexing, the PMOO analysis suffers from the segregation effect [41], both flows assume to only receive service after the respective other flow was forwarded by server s_1 – an unattainable pessimistic forwarding scenario in the analysis-internal view on the network. FP tries to steer the analysis such that it does not have to apply this pessimism by prolonging flows inside the analysis: the dashed lines in Figure 2(b), depict potential prolongations of the two flows' paths. Each prolongation alternative that matches their sinks will allow for their aggregate treatment at s_1, mitigating the problem. Yet, this adds interference to the foi. Therefore, we search for the best prolongation alternative trading off both

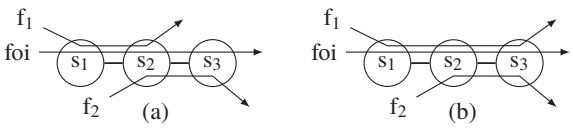

Figure 3: (a) Tandem network and (b) its prolonged version

aspects. This search approach does not scale, neither are there hopes that PMOO delay bounds improve much [11].

IV. Flow Prolongation in the NC FIFO Analyses

In this Section, we address the question of how flow prolongation can improve the NC-derived worst-case delay bound for a flow of interest in the FIFO analysis.

In the PMOO analysis, demultiplexing is the dominant problem that causes a loss of tightness. While the problem of demultiplexing applies to the LUDB analysis for FIFO networks, too, it suffers from yet another and more impactful problem that we address with flow prolongation: the lack of the PMOO property. To implement the property, an analysis needs to first create an end-to-end view on a tandem. LUDB, and thus the DEBORAH tool, cannot achieve this for non-nested interference patterns (see Figure 3(a)). It can only analyze nested tandems in a PMOO fashion where cross-flow paths do not overlap.

To apply LUDB nonetheless, the tandem is cut into a sequence of sub-tandems with nested interference patterns. In Figure 3, the tandem can be cut before or after server s_2. Either alternative has the very same drawback: a cross-flow is cut, too, and to get it onto the subsequent sub-tandem, an explicit bound on its arrivals has to be computed. This is achieved with the deconvolution $\oslash$ (see Appendix B) or Theorem 2 in Section IV-A, adding the cross-flow's original burst term to the analysis once more – PMOO is not achieved. Let server s_i provide service β_i and let flow f_j put α_j data into the network. The respective (min,plus)-analysis terms using $\oslash$ as derived by DEBORAH which bound the foi's delay are:

$$h(\alpha_{\text{foi}}, (\beta_1 \ominus_\theta \alpha_1) \otimes (((\beta_2 \ominus_\theta (\alpha_1 \oslash (\beta_1 \ominus_\theta \alpha_{\text{foi}}))) \otimes \beta_3) \ominus_\theta \alpha_2)) \tag{3}$$

for the cut left of s_2 and for the cut right to it:

$$h(\alpha_{\text{foi}}, ((\beta_1 \otimes (\beta_2 \ominus_\theta \alpha_2)) \ominus_\theta \alpha_1) \otimes (\beta_3 \ominus_\theta (\alpha_2 \oslash (\beta_2 \ominus_\theta ((\alpha_{\text{foi}} + \alpha_1) \oslash \beta_1))))). \tag{4}$$

Curves and binary (min,plus)-operations are defined in Appendix A. For this example, it is already sufficient to note that every occurrence of the deconvolution $\oslash$ reduces the tightness of the computed delay bound.

In this paper, we devise an alternative strategy to create a tandem with nested interference only and thus less cuts, less occurrences of $\oslash$ and more PMOO property implementation: flow prolongation. By prolonging cross-flow f_1 in this small sample tandem by another hop, we create the one shown in Figure 3(b). Without overlapping interference, the foi's DEBORAH-derived delay bound becomes:

$$h(\alpha_{\text{foi}} + \alpha_1, \beta_1 \otimes ((\beta_2 \otimes \beta_3) \ominus_\theta \alpha_2)) \tag{5}$$

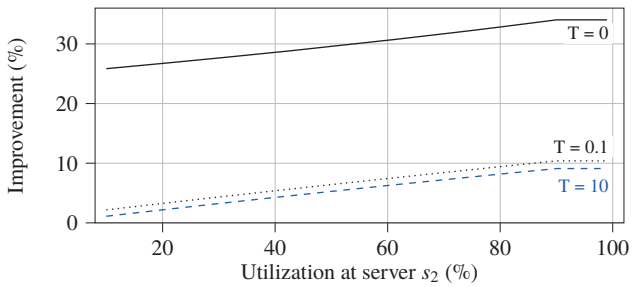

Figure 4: Delay bound improvements by using FP in the DEBORAH analysis of the network in Figure 3

By its lack of deconvolutions, i.e., single appearances of each involved flow's arrival curve, the term clearly shows that the PMOO principle is implemented. Yet, at the expense of aggregating the foi with its cross-flow f_1[1]. We have tested this new instantiation of flow prolongation to improve the DEBORAH-derived LUDB bounds for different curve settings in the network shown in Figure 3. Service curves were set to $\beta_{R=30,T}$ and arrival curves to $\gamma_{r=\frac{u}{10},0.1}$ where u denotes the utilization $\frac{3r}{R}$ at the server that always sees three flows, s_2, and varying latencies. Note, that our setting guarantees for finite delay bounds. Figure 4 shows the results.

FP for the DEBORAH analysis, henceforth called DEBORAH-FP, is a very promising approach to implement the PMOO property in the algebraic NC analysis. Its application vastly differs from the PMOO analysis in arbitrary multiplexing. Put simple, the necessary preconditions for FP to have a positive impact on each analysis are as follows:

- For the PMOO analysis, prolong cross-flows that start at *the same* server to the same last server.
- For the DEBORAH analysis, prolong cross-flows that start at *different* servers to the same last server.

A. DEBORAH in feed-forward FIFO networks

For the analysis of feed-forward FIFO networks, we integrated DEBORAH into the NetworkCalculus.org Deterministic Network Calculator (NCorg DNC) [44] as its feed-forward analysis already provides the required decomposition of the network into a sequence of tandems [45]. Second, LUDB only computes delay bounds but we can use DEBORAH for bounding arrivals of cross-traffic by using the following theorem, an alternative to the deconvolution-based computation:

Theorem 2 (Output From Delay [46]): Consider a tandem of servers $\mathcal{T}$ that offers a service curve β. Assume flow f with arrival curve α traverses $\mathcal{T}$, experiencing a delay bounded by d. Then $\alpha'(t) = \alpha(t + d)$ bounds the output of f from $\mathcal{T}$.

The impact of our contribution does not rely on this rather inaccurate bounding technique. We put flow prolonged

[1]DEBORAH can only work with a single flow (aggregate) per distinct path on the tandem. Input to DEBORAH needs to be formatted accordingly. In case a cross-flow has the same path as the foi we thus get an aggregate delay bound instead of computing a residual service curve for the foi – an alternative derivation of a valid upper delay bound that now implements PMOO, too. Either is subject to overly pessimistic interference assumptions.

tandems into the DEBORAH tool that applies a more refined computation internally if the tandem is non-nested. Yet, DEBORAH does not expose this computation to the user.

A recent overview on further NC tools can be found in [47]. We also investigated another tool[2] for the analysis of FIFO networks, which uses a linear program (LP) to compute the delay bound. Our evaluations showed that it scaled insufficiently for inclusion in our numerical evaluation, even on small networks with 20 flows, mainly due to the large number of LP constraints generated, confirming previous results [9].

B. The Challenge to Apply Flow Prolongation

As mentioned in Section I and illustrated in Figures 1(a) and 2, on each tandem of length n with m cross-flows, FP may explore $O(n^m)$ prolongation alternatives. It was shown for arbitrary multiplexing that exhaustive FP analysis does not scale in feed-forward networks [11]. Due to their similarity, the scaling problem also holds when applying DEBORAH-FP.

1) Restricting the Application of FP: The most straightforward trade-off between delay bound tightness and computational complexity is, of course, to restrict the use of FP inside the NC analysis. We deviate from an exhaustive use of FP on every tandem to a selective use where it has the most impact on the delay bound. It turned out that this is achieved by only applying FP to the analysis of the foi, not for bounding the arrivals of its cross-flows. This creates the FP_{foi} variants PMOO-FP_{foi} and DEBORAH-FP_{foi}. Figure 5 illustrates the delay bound gap between PMOO-FP and PMOO-FP_{foi}, and between DEBORAH-FP and DEBORAH-FP_{foi}, namely:

$$delay\ bound\ gap = \frac{delay_{\text{foi}}^{\text{FP}_{\text{foi}}} - delay_{\text{foi}}^{\text{FP}}}{delay_{\text{foi}}^{\text{FP}}} \tag{6}$$

For more than 99 % of the studied flows, the delay bound is unchanged. On average, the relative error is only of 0.58 % for PMOO and 1.18 % for DEBORAH. Those values illustrate that the loss of tightness of using FP_{foi} instead of FP is minimal.

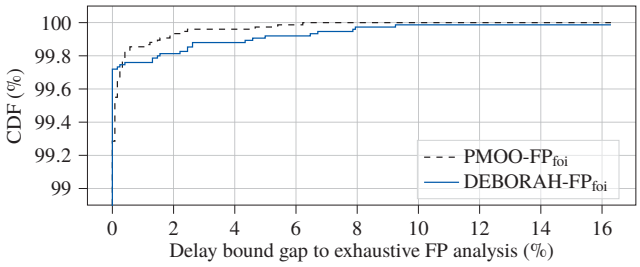

Figure 5: Delay bound gap of FP_{foi} analyses based on the evaluation dataset presented in Section V-D

In order to see the impact on the execution time of running flow prolongations only on the foi's analysis, Figure 6 illustrates the relative execution time of FP against FP_{foi}, namely:

$$Relative\ execution\ time = \frac{Execution\ time\ FP}{Execution\ time\ FP_{foi}} \tag{7}$$

[2]https://github.com/annebouillard/NetCalBounds based on [24, 25]

PMOO-FP_{foi} is 1.3 times faster than PMOO-FP in average, while there is almost no difference in execution time between DEBORAH-FP and DEBORAH-FP_{foi}.

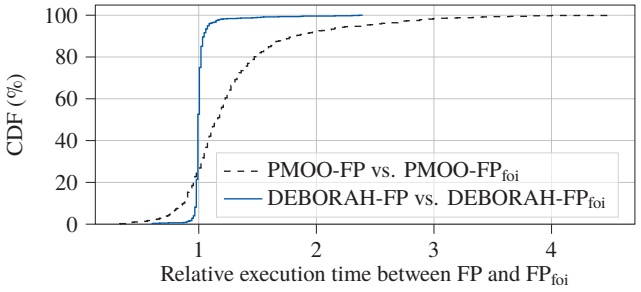

Figure 6: Relative execution time of a flow's analysis based on the evaluation dataset presented in Section V-D

2) PMOO-FP's explored alternatives: We can reasonably reduce the use of FP to a single tandem. On this tandem, the amount of prolongation alternatives to explore can be further reduced. In practice, not all cross-flows go over only the first server such that they can also be prolonged to any following one. Moreover, PMOO-FP already cuts out all those alternatives that cannot impact the analysis by circumventing the need to carry over demultiplexing – as described in the necessary FP precondition above. Similarly, we improved DEBORAH-FP to not prolong if there is no potential to convert a non-nested interference pattern to a nested one. Still, the amount of prolongation alternatives for the dataset evaluated in this paper is forbiddingly large, see Figure 7. Note for a large number of cross-flows, networks may have been excluded from this preliminary evaluation due to a 1 hour deadline set for computing data. As expected, we get an exponential scaling between the number of cross-flows and the number of explored alternatives.

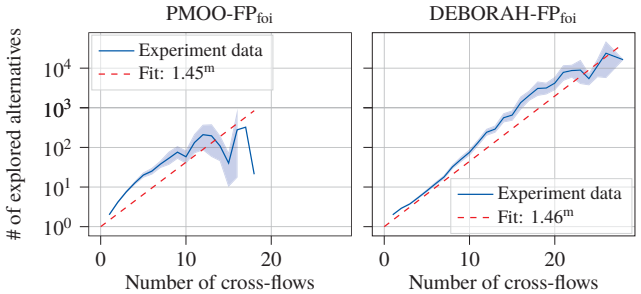

Figure 7: Relation between the number of cross-flows and the number of explored prolongation alternatives by PMOO-FP_{foi} and DEBORAH-FP_{foi}

Overall, we need a better way to find the best prolongation alternative that improves the delay bound to be derived. In this paper, we propose DeepFP that can be trained on either a PMOO-FP_{foi} or a DEBORAH-FP_{foi} dataset to predict the best alternative(s). We show that DeepFP makes the FP feature scale, that PMOO-FP_{foi} provides only minor improvements

over PMOO and that, in contrast, FP has a considerable impact on the LUDB analysis when coupled with its implementation, DEBORAH, to DEBORAH-FP$_{\text{foi}}$.

V. Effective FP Predictions with a GNN

We make the FP analysis scale with GNN predictions and show that the impact vastly depends on the multiplexing assumption. We develop our universal DeepFP heuristic in this section, based in part on the work proposed in DeepTMA [13, 14]. As illustrated in Figure 1 and Algorithm 1, the main intuition behind DeepFP is to avoid the exhaustive search for the best prolongation by limiting it to a few alternatives. The heuristic's task is then only to predict the best flow prolongations, which are then fed to the NC analysis. This ensures that the bounds provided are formally valid.

Algorithm 1 DeepFP analysis of network $\mathcal{N}$ and flow f_{foi}

$\mathcal{G}$:= graphTransformation($\mathcal{N}$, f_{foi}) → see Algorithm 2
$prolongations$:= GNN($\mathcal{G}$) → see Section V-A
$\mathcal{N}_p$:= networkWithFlowProlongations($\mathcal{N}$, $prolongations$)
return Network Calculus analysis of $\mathcal{N}_p$ and f_{foi}

For DeepFP, we used a Graph Neural Network (GNN) as heuristic, since it was shown in DeepTMA to be a fast and efficient method. 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. Our heuristic transforms the networks into graphs, which are processed by the GNN. The output of the GNN is then fed to PMOO-FP$_{\text{foi}}$ or DEBORAH-FP$_{\text{foi}}$, which finally performs the NC analysis on the subset of combinations suggested by the GNN.

A. Graph Neural Networks

As for DeepTMA, we use the framework of GNNs introduced in [28, 29]. They are a special class of neural networks for processing graphs and predict values for nodes or edges depending on the connections between nodes and their properties. The idea behind GNNs is called *message passing*, where so-called *messages* – i.e., vectors of numbers $\mathbf{h}_v \in \mathbb{R}^k$ – are iteratively updated and passed between neighboring nodes. Those messages are propagated throughout the graph using multiple iterations. We refer to [48] for a formalization of many concepts recently developed around GNNs.

As with DeepTMA, we selected Gated Graph Neural Networks (GGNN) [49] for our model, with the addition of edge attention. For the edge attention mechanism, we selected an approach similar to [50], where each edge (u, v) in the input graph is weighted with a parameter $\lambda_{(u,v)} \in (0, 1)$, such that:

$$\lambda_{(u,v)}^{(t)} = \sigma\left(FFNN\left(\left\{\mathbf{h}_u^{(t)}, \mathbf{h}_v^{(t)}\right\}\right)\right) \tag{8}$$

with $FFNN$ a feed-forward neural network $\mathbf{h}_v^{(t)}$ representing the message from node v at iteration t, σ the sigmoid function, and $\{\cdot, \cdot\}$ the concatenation. In summary, the hidden node update function becomes:

$$\mathbf{h}_v^{(t)} = GRU\left(\mathbf{h}_v^{(t-1)}, \sum_{u \in \text{NBR}(v)} \lambda_{(u,v)}^{(t-1)} \mathbf{h}_u^{(t-1)}\right) \tag{9}$$

with $\text{NBR}(v)$ of v the set of neighbors of node v, and GRU a Gated Recurrent Unit (GRU) [51].

B. Model transformation

Since we work with a machine learning method, we need an efficient data structure for describing a NC network which can be processed by a neural network. We chose undirected graphs, as they are a natural structure for describes networks and flows. Due to their dynamic sizes, networks of any sizes may be analyzed using our method.

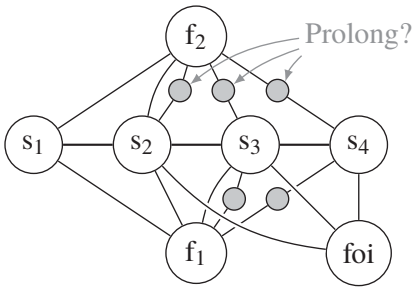

Figure 8: Graph encoding of the network from Figure 2(a)

We follow Algorithm 2 for this graph transformation, also illustrated and applied in Figure 8 on the network from Figure 2(a). Each server is represented as a node in the graph, with edges corresponding to the network's links. The features of a server node are its service curve parameters, namely its rate and latency. Each flow is represented as a node in the graph, too. The features of a flow node are its arrival curve parameters, namely its rate and burst. Additionally, the foi receives an extra feature representing the fact that it is the analyzed flow.

Algorithm 2 Graph transformation of network $\mathcal{N}$ for flow f_{foi}

$\mathcal{G}$:= empty undirected graph
for all server s_i **in** network $\mathcal{N}$ **do** $\mathcal{G}$.addNode(s_i)
for all link (s_i, s_j) **in** network $\mathcal{N}$ **do** $\mathcal{G}$.addEdge(s_i, s_j)
for all flow f_i **in** network $\mathcal{N}$ **do**
  $\mathcal{G}$.addNode(f_i)
  for all server s_j **in** f_i.path() **do** $\mathcal{G}$.addEdge(f_i, s_j)
for all flow f_i **in** network $\mathcal{N}$ excluding f_{foi} **do**
  for all server s_j **in** f_{foi}.path() **do**
    if prolongation $P_{f_i}^{s_j}$ of flow f_i to s_j is valid **then**
      $\mathcal{G}$.addNode($P_{f_i}^{s_j}$)
      $\mathcal{G}$.addEdges((f_i, $P_{f_i}^{s_j}$), ($P_{f_i}^{s_j}$, s_j))
return $\mathcal{G}$

To encode the path taken by a flow in this graph, we use edges to connect the flow to the servers it traverses. Compared

to the original DeepTMA graph model from [13], we simplify one aspect: we do not include path ordering nodes that tell us the order of servers on a crossed tandem. DeepTMA was shown to benefit only marginally from the effort to incorporate this additional information [14] and we confirmed the same behavior in preliminary DeepFP numerical evaluations.

To represent the flow prolongations, *prolongation* nodes ($P_{f_i}^{s_j}$) connecting the cross-flows to their potential prolongation sinks are added to the graph. Those nodes contain the hop count according to the foi's path as main feature – this is sufficient to later feed the prolongation into the NC analysis, path ordering nodes are not required for this step either.

The last server of a cross-flow's unprolonged path is also represented as a node (s_3 for f_1 and s_2 for f_2 in Figure 8). Those nodes represent the choice to not prolong a flow.

Based on this graph representation, we define two classification problems for the neural network. The first one is decide if it is worthwhile to apply the prolongation algorithm or not. For this, we use a binary classification of the foi node.

The second classification problem is to decide where to prolong the flows if necessary, by applying a binary classification on the prolongation nodes. Namely for each cross-flow f and each potential sink s, the neural network assigns a score $P_{f,s}$ between 0 and 1 to the corresponding prolongation node. For each flow, the prolongation node with the highest score decides which sink to use for prolonging the flow. As illustrated in Figure 1(b), those predictions are then fed to PMOO-FP$_{\text{foi}}$ or DEBORAH-FP$_{\text{foi}}$, which finally performs the NC analysis. Since the GNN might also choose not to prolong, the standard PMOO or DEBORAH analyses are only performed if explicitly requested by the GNN.

C. Implementation

We implemented the GNN used in DeepFP using PyTorch [52] and pytorch-geometric [53]. Optimal parameters for the neural network size and the parameters for training were found using hyper-parameter optimization. Table I illustrates the size of the GNN used for the evaluation in Section VI.

Layer	NN Type	Size
init	FFNN	$(11, 96)_w + (96)_b$
Memory unit	GRU cell	$(96, 96)_w + 2 \times \{(288, 96)_w + (96)_b\}$
Edge attention	FFNN	$(192, 96)_w + (96)_b + (192, 96)_w + (1)_b$
out hidden layers	FFNN	$2 \times \{(96, 96)_w + (96)_b\}$
out final layer 1	FFNN	$(96, 1)_w + (1)_b$
out final layer 2	FFNN	$(96, 1)_w + (1)_b$
	Total:	104 455 parameters

Table I: Size of the GNN used in Section VI. Indexes represent respectively the weights (w) and biases (b) matrices

D. Dataset generation

To train our neural network architecture using a supervised learning method, we generated a set of random tandem topologies (as to check the FP preconditions of Section IV). For each created server, a rate-latency service curve was generated with uniformly random rate and latency parameters. A random number of flows was generated with random source and sink servers. 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.

For each generated topology, the NCorg DNC v2.6.1 [44] is then used for analyzing each flow and record the different iterations of PMOO-FP$_{\text{foi}}$ and DEBORAH-FP$_{\text{foi}}$. Namely we extract the combinations of flow prolongations which resulted in the lowest end-to-end delay during the exhaustive search. Each analysis is run with a maximum deadline of 1 hour.

We extended the NCorg DNC tool to integrate the DEBORAH tool. For our evaluations, we run DEBORAH in so-called STA mode (Single Tandem Analysis) [16] instead of the default MSA (Multiple Sub-tandem Analysis). The MSA mode computes per-sub-tandem delay bounds and adds them up. In this mode, DEBORAH cannot be used to implement the PMOO principle, not even the PBOO one. Yet, STA has a worse execution time since more variables have to be optimized simultaneously.

Since PMOO-FP$_{\text{foi}}$ and DEBORAH-FP$_{\text{foi}}$ may not bring any benefits compared to PMOO or DEBORAH, either due to no alternatives for prolonging flows or no end-to-end delay improvement by any alternative, we restrict the dataset to networks and flows where FP is applicable (i.e., flows with prolongation options). Table II contains statistics about the generated dataset. In total approximately 54 000 flows were generated and evaluated for the training dataset, and 10 000 for the numerical evaluation presented in Section VI. The dataset is available online[3] to reproduce our learning results.

Parameter	Min	Max	Mean
# of servers	4	10	7.8
# of flows	5	35	24.5
# of cross-flows	1	21	4.1
# of prolong. comb. (PMOO-FP$_{\text{foi}}$)	2	4024	16.8
# of prolong. comb. (DEBORAH-FP$_{\text{foi}}$)	2	131072	247.1
Flow path length	3	9	4.1
Number of nodes in graph	11	128	43.3

Table II: Statistics about the generated dataset

E. Neural network training

We use standard gradient descent techniques to train our GNN, using the binary cross-entropy loss function, namely the optimization goal is to minimize:

$$loss(T, P) = \sum_{f \in \mathcal{F}, s \in \mathcal{S}_f} \left(T_{f,s} \log P_{f,s} + (1 - T_{f,s}) \log (1 - P_{f,s})\right) \tag{10}$$

with T_f^s representing the target score for the prolongation, with 1 if it is selected for prolongation and 0 otherwise.

We follow a standard supervised learning approach for training the neural network. Since the choice of flow prolongations may have multiple equally-optimal solutions, the choice of which target solution to provide as training data for the neural network is not obvious. In other words, the target vector $T_{f,s}$ in Equation (10) has to be defined according to a single

[3] https://github.com/fabgeyer/dataset-rtas2021

solution, but multiple equally good solutions are available. In our experiments, training the GNN on a single solution resulted in poor convergence of the model.

To provide target vectors which enable the neural network to be trained efficiently, we use here a concept inspired by hindsight loss [54, 55]. We dynamically find the correct target vector T which is the closest to the predicted score by the neural network and use it in the loss function. The loss function introduced in Equation (10) becomes:

$$\mathcal{L} = \min_{T \in \tau_{opt}} loss(T, P) \tag{11}$$

where τ_{opt} is the set of flow prolongation choices leading to an optimal solution.

Since we address two FP instantiations that are even orthogonal as seen in their preconditions to have a positive impact, we define two versions of DeepFP as PMOO-DeepFP$_{\text{foi}}$ and DEBORAH-DeepFP$_{\text{foi}}$. The same graph representation and features are used regardless of the NC analyses, but two different training processes and resulting trained weights of the GNN are produced.

F. Flow prolongation choices

To improve the outcome of a DeepFP analysis at a small computational cost, we propose here to use the prediction vector of the GNN to generate multiple flow prolongation combinations. Those combinations are then analyzed using the NCorg DNC and the combination leading to the lowest end-to-end delay is kept. We name this extension DeepFP$_k$ where k corresponds to the number of combinations generated.

First, we consider the prediction vector of the GNN as a vector of probabilities of where to prolong flows. A categorical distribution parameterized by those probabilities is generated for each cross-flow and used to generate the k combinations.

This first version of DeepFP$_k$ does not make use of the expert knowledge mentioned in Section IV-B2, where some combinations are excluded from PMOO-FP$_{\text{foi}}$'s and DEBORAH-FP$_{\text{foi}}$'s exhaustive searches since they are known to be of lower quality than other combinations. In other words, the GNN can choose a combination of flow prolongations which might not have been explored by PMOO-FP$_{\text{foi}}$'s or DEBORAH-FP$_{\text{foi}}$'s exhaustive search. This reflects the generally observed wish to apply machine learning to a dataset without becoming an expert in the domain and without tailoring the dataset to learn from accordingly.

Last, we describe a second extension of DeepFP, called DeepFP$^+$ which is able to use this expert knowledge. In order to avoid selecting those excluded combinations, we define a matrix of explored combinations $\mathcal{C}$ containing their target vectors, with dimensions $(\#\ of\ combinations, \#\ of\ prolongation\ nodes)$. Using the prediction vector $P_{f,s}$ from the GNN, we compute a vector containing a score for each combination:

$$CombinationScores = \mathcal{C} \times \left[P_{f_i,s_k} \cdots P_{f_j,s_l}\right]^T \tag{12}$$

To generate k combinations, we select the top-k combinations having the best scores in the $CombinationScores$ vector.

We numerically evaluate later in Sections VI-B and VI-D the impact of DeepFP$^+$ in terms of tightness and additional execution time of enumerating the combinations used by PMOO-FP$_{\text{foi}}$ and DEBORAH-FP$_{\text{foi}}$.

VI. Numerical evaluation

Our numerical evaluation aims to answer two questions:

1) How much delay bound improvement can FP achieve?
2) How well does the GNN predict the FP alternative?

As FP does not scale well and we therefore proposed DeepFP in the first place, both aspects are naturally intertwined.

In the following, we show details about DeepFP performance in terms of tightness as well as execution time. Improvements in both will directly be applicable to and have an impact on any real-world application of the NC methodology. In order to illustrate the benefits of DeepFP, we also do a comparison against a heuristic which randomly selects one or multiple prolongation alternatives – a low-effort, non-expert alternative to add FP to an analysis. We label this heuristic as RND_k in this section, with k being the number of random alternatives evaluated. At first, we use DeepFP without the extension using additional expert knowledge described in Section V-F. All evaluations presented here were done with the evaluation dataset described in Section V-D, except for Section VI-C which used larger networks.

A. Accuracy and delay bound gap

To quantitatively evaluate the performance of our approach, we use the relative gap between the delay bound given by PMOO-FP$_{\text{foi}}$ and DEBORAH-FP$_{\text{foi}}$ and the delay bound given by a heuristic, incl. the non-FP original analysis:

$$delay\ bound\ gap^{\text{FP}}_{\text{foi}} = \frac{delay^{\text{heuristic}}_{\text{foi}} - delay^{\text{FP}_{\text{foi}}}_{\text{foi}}}{delay^{\text{FP}_{\text{foi}}}_{\text{foi}}} \tag{13}$$

A value of $delay\ bound\ gap^{\text{FP}}_{\text{foi}}$ close to zero indicates that the heuristic produced a tight result compared to the exhaustive search. Larger values indicate that the heuristic chose a bad prolongation, i.e. the bound is loose.

The results are shown in Figure 9. First to note is that FP does not have a significant impact in PMOO – we confirm the finding of [11] in a larger evaluation by observing an average gap between PMOO-FP$_{\text{foi}}$ and PMOO of just 3.7 %. Neither the random heuristic nor DeepFP can thus achieve a considerable delay bound improvement, although the predictions taken are very accurate.

For DEBORAH-FP, we can report a completely different picture. Having brought the FP property to the DEBORAH analysis had a huge impact on the delay bound tightness. We see that an average gap of 60.75 % between DEBORAH and DEBORAH-FP$_{\text{foi}}$ analysis results was opened when adding the exhaustive FP$_{\text{foi}}$ feature. Moreover, reducing the effort by random selection of prolongation alternatives did not perform well, even RND$_{16}$ leaves an average gap of 11.68 %. On the other hand, our DeepFP closes this gap successfully. Even the version with a single prediction pushes the gap down to 2.57 %

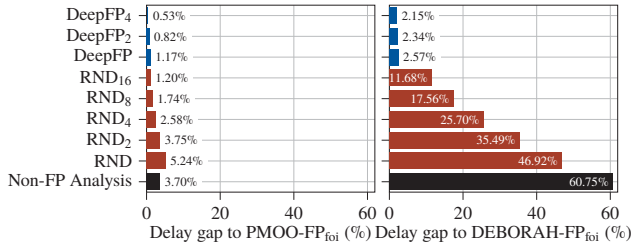

Figure 9: Average delay bound gap of heuristics against PMOO-FP$_{\text{foi}}$ and DEBORAH-FP$_{\text{foi}}$

such that an increase of proposed prolongation alternatives does not have a big impact anymore.

More detailed results are presented in Figure 10, where we illustrate the delay bound gap of DeepFP, the random heuristic and standard PMOO or DEBORAH analyses, confirming our findings that DEBORAH-FP$_{\text{foi}}$ is a big improvement over DEBORAH and DeepFP is the key to its efficient application.

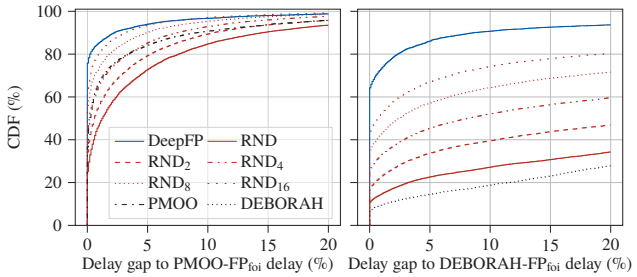

Figure 10: Delay bound gap of heuristics against PMOO-FP$_{\text{foi}}$ and DEBORAH-FP$_{\text{foi}}$

The accuracy of DeepFP is shown in Figure 11. For each analyzed flow in the test dataset, the method is accurate if the computed end-to-end delay bound is equal to the best end-to-end delay bound computed by PMOO-FP$_{\text{foi}}$ or DEBORAH-FP$_{\text{foi}}$. In average, DeepFP is able to predict the correct prolongation for 69.6 % of the flows for PMOO-FP$_{\text{foi}}$ and 60.9 % for DEBORAH-FP$_{\text{foi}}$. Generating multiple combinations as introduced in Section V-F increases the accuracy to 75.3 % and 64.3 % respectively for $k = 4$. In comparison, the random heuristic with one choice achieves only 14.5 % and 8.7 % respective accuracy. DeepFP is making more reasonable, more accurate predictions.

B. Impact of additional expert knowledge for DeepFP$^+$

We introduced DeepFP$^+$ in Section V-F, an extension of DeepFP making additional use of expert knowledge to explicitly filter out prolongation combinations which are known to be of lower quality. We numerically compare DeepFP and DeepFP$^+$ in Figures 12 and 13. As expected, DeepFP$^+$ is able to achieve a better accuracy for both PMOO-FP$_{\text{foi}}$ and DEBORAH-FP$_{\text{foi}}$. Nevertheless, while the delay bound gap of DeepFP$^+$ to the exhaustive search of DEBORAH-FP$_{\text{foi}}$ is indeed reduced compared to DeepFP, DeepFP achieves better

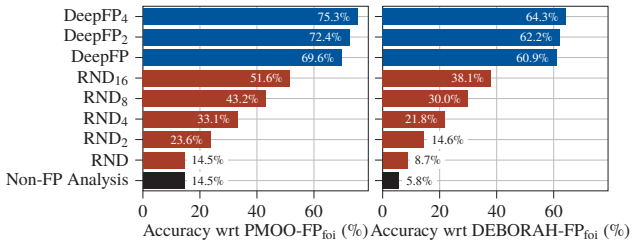

Figure 11: Accuracy of DeepFP, the random heuristic, and the non-FP analyses

Parameter	Min	Max	Mean
# of servers	2	16	8.7
# of flows	5	254	162.3
Flow path length	1	16	3.2

Table III: Statistics about the larger generated dataset

results than DeepFP$^+$ for PMOO-FP$_{\text{foi}}$. This means that the expert knowledge of reducing the state of possible solutions might not be necessary.

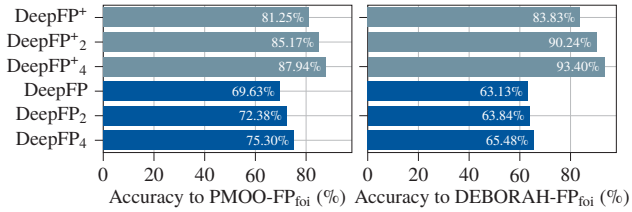

Figure 12: Accuracy of DeepFP$^+$ and DeepFP

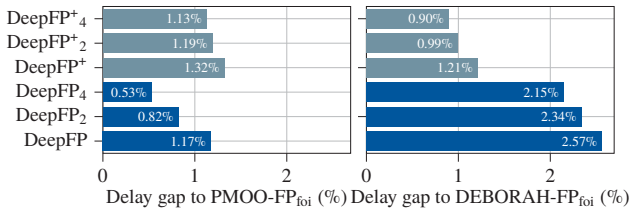

Figure 13: Average delay bound gap of DeepFP$^+$ and DeepFP

The additional computational cost of using DeepFP$^+$ will be evaluated later in Figure 17.

C. Scalability on larger networks

To evaluate the scalability of our approach with respect to the network size, we also evaluated DeepFP on networks with a larger number of servers and flows. The same random network generator as for the training dataset is used, but the number of servers and flows is scaled to larger values. Statistics about this additional dataset are presented in Table III. The training data used for the GNN is unchanged, namely we still restrict it to the smaller networks introduced in Section V-D and Table II.

The results of the exhaustive PMOO-FP$_{\text{foi}}$ and DEBORAH-FP$_{\text{foi}}$ are not available here due to their too long execution time, taking multiple days per network to compute in some cases. Instead, we use here the standard PMOO and DEBORAH analyses in order to evaluate the gain in tightness of using DeepFP. As in Equation (13), we define the delay bound gap to PMOO and DEBORAH (i.e. the analyses without the FP property) as:

$$delay\ bound\ gap_{\text{foi}}^{\text{non-FP}} = \frac{delay_{\text{foi}}^{\text{non-FP}} - delay_{\text{foi}}^{\text{heuristic}}}{delay_{\text{foi}}^{\text{non-FP}}} \quad (14)$$

A large positive value of $delay\ bound\ gap_{\text{foi}}^{\text{non-FP}}$ indicates that the heuristic with the FP property gained tightness over the standard PMOO or DEBORAH analysis. In the opposite, a negative value indicates that the bound is less tight.

Numerical results are summarized in Figure 14. For the PMOO analysis, the random heuristic results in a negative delay bound gap in average, namely the resulting delay bounds are worse than by simply using the standard PMOO analysis, even for the larger values of $k = 32$. Despite this, DeepFP is able to achieve an average gain in tightness of 1.06 % for PMOO. For the DEBORAH analysis, the random heuristic results in a gain in tightness of only 0.25 %, where DeepFP is able to achieve a gain of 13.74 %.

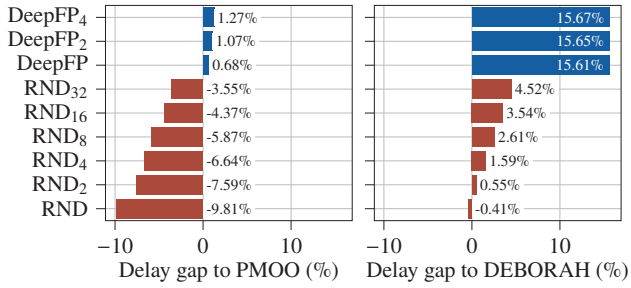

Figure 14: Average delay bound gap of DeepFP to standard PMOO and DEBORAH on the larger networks

Overall, these results illustrate that a simple random choice is not sufficient to improve tightness using flow prolongations. DeepFP is able to accurately choose flow prolongations resulting in a gain in tightness, even on larger networks than the ones it is was trained on.

D. Execution time

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 analyze all its flows, without including the startup time or the time taken for initializing the network data structures. The execution times were measured on a server with dual AMD EPYC 7542 CPU. The GNN was executed using GPU acceleration with a Nvidia GTX 1080 Ti, while the NC analysis is still executed on CPU. No batching was used, i.e. the GNN analyzes one network at a time.

We first illustrate the average relative execution time of the FP analyses against the non-FP analysis in Figure 15, namely:

$$\frac{Execution\ time\ FP}{Execution\ time\ non\text{-}FP} \quad (15)$$

This measure helps us understand the cost of using FP. In average, DeepFP with GPU acceleration is approximately an order of magnitude faster than PMOO-FP$_{\text{foi}}$, and almost three orders of magnitude faster than DEBORAH-FP$_{\text{foi}}$. Taking into account the tightness of the method illustrated earlier in Figure 9, those results show that DeepFP is able to achieve a good balance between tightness and computational cost.

DeepFP without GPU acceleration is approximately an order of magnitude faster than DEBORAH-FP$_{\text{foi}}$, making it still an appealing solution despite it's slower execution time. In the case of PMOO-FP$_{\text{foi}}$, DeepFP is actually slower than the exhaustive analysis.

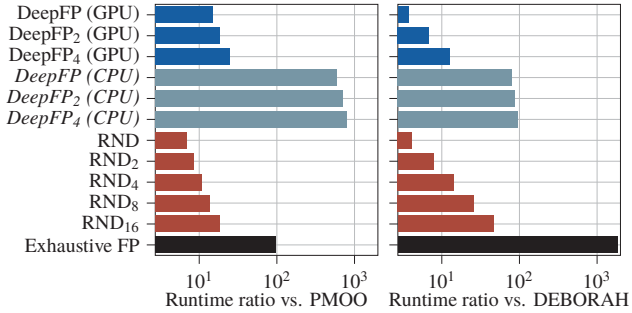

Figure 15: Average relative execution time of different analyses

Second, we evaluate the execution time of the GNN in comparison to the total execution time of the analysis. We use the following measure:

$$\frac{Execution\ time\ GNN}{Total\ execution\ time\ (GNN + NC)} \quad (16)$$

Results are presented in Figure 16. When taking advantage of the GPU acceleration, the GNN prediction takes 17.2 % in average of the analysis for PMOO-DeepFP$_{\text{foi}}$, and 2.46 % in average for DEBORAH-DeepFP$_{\text{foi}}$.

Without GPU acceleration, the GNN prediction takes 91.4 % in average of the analysis for PMOO-DeepFP$_{\text{foi}}$, and 64.3 % in average for DEBORAH-DeepFP$_{\text{foi}}$. From Figures 9 and 16, we conclude that DeepFP is mostly attractive in case GPU acceleration is used for the GNN. Despite this drawback, we note that various techniques may be used to speed-up neural network inference on CPU, such as by reducing the size of the GNN, or using mixed-precision floats.

Finally, we evaluate the execution time of the additional enumeration of prolongation combinations used by DeepFP$^+$. As for the GNN part, we use the following measure:

$$\frac{Execution\ time\ Enum.}{Total\ execution\ time\ (Enum. + GNN + NC)} \quad (17)$$

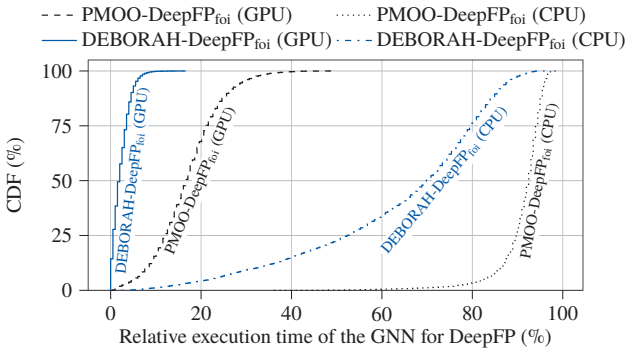

Figure 16: Relative execution time of the GNN for DeepFP

Results are presented in Figure 17. In average, the enumeration of prolongation combinations takes 7.22 % of the execution time for PMOO-DeepFP$^+$, and 4.17 % for DEBORAH-DeepFP$^+$. This illustrates that the gains in tightness of DeepFP$^+$ can be achieved at a small computational cost.

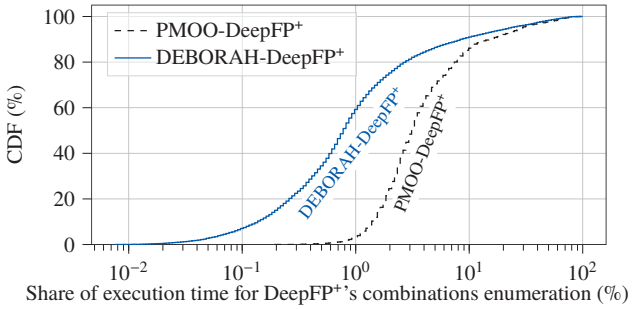

Figure 17: Relative execution time of the GNN for DeepFP

E. Feature importance and sensitivity analysis

We perform here a sensitivity analysis of the choices of flows prolongation of PMOO-FP$_{foi}$ and DEBORAH-FP$_{foi}$ to better understand which parameters influence the decision for the best combination. To numerically evaluate this, we randomly modify the curve parameters $p_{original}$ with a relative scale ϵ according to the following uniform distribution:

$$p_{new} \sim \mathcal{U}\left(p_{original}(1-\epsilon), p_{original}(1+\epsilon)\right) \qquad (18)$$

We then compare the share of flows where the best combination of flows prolongation have changed due to the random change of curves parameters.

Results are presented in Figure 18. We note that the server's rate has the largest impact on the choice of flows prolongation. Arrival curve parameters also impact also the flows prolongations, but with less magnitude than the service rate. Finally, the service latency has almost no influence on the choice of prolongations, where even large changes of its value result in less than a 1 % change for arbitrary multiplexing, or no changes at all for DEBORAH-FP$_{foi}$. The service latency, in

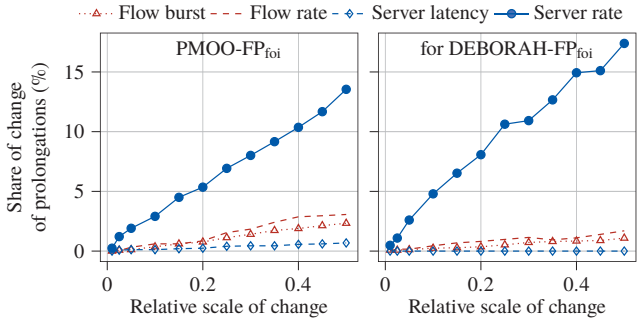

Figure 18: Sensitivity analysis of the NC analyses

contrast, is an additive factor in the residual forwarding service computation, making it considerably less impactful.

We use the permutation-based importance measure [56, 57] in order to assess each feature's importance for DeepFP. For each input feature presented in Section V-C, we randomize it by randomly permuting its values in the evaluation set, and assess the impact it has on the relative error of the predictions. We define the feature importance as:

$$delay\ bound\ gap_{\text{foi}}^{Feature} - delay\ bound\ gap_{\text{foi}}^{Baseline} \qquad (19)$$

with $delay\ bound\ gap_{\text{foi}}^{Baseline}$ corresponding to the delay bound gap of DeepFP without column permutation.

Results are presented in Figure 19. As expected from the sensitivity analysis, the server rate is the feature having the largest impact on the prediction of the GNN. The other features have almost two orders of magnitude less importance.

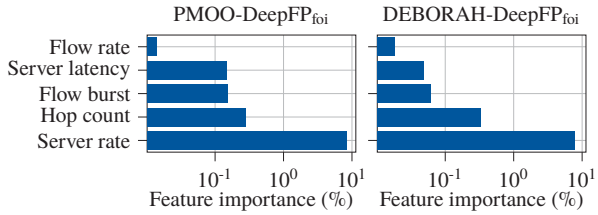

Figure 19: Feature importance of DeepFP

VII. Conclusion

We introduced DeepFP in this paper, an approach for making the NC analysis feature Flow Prolongation scale. FP can be paired with either of the two predominant flow multiplexing assumptions, arbitrary or FIFO, and we show that it is most impactful when bounding the flow of interest's delay (compared to bounding cross-flow arrivals). As each multiplexing assumption's analysis must be trained differently, we devise two analyses: PMOO-FP$_{foi}$ for arbitrary multiplexing and DEBORAH-FP$_{foi}$ for FIFO multiplexing. The latter is based on the novel insight that FP can improve the implementation of the PMOO property in the current LUDB FIFO analysis and thus its tool DEBORAH. Our numerical

results show considerably tighter delay bounds of this state-of-the-art algebraic NC FIFO analysis, the average gap to the classic non-FP delay bound rises to 60.75 % – yet at the expense of computational effort. DeepFP predictions solve this problem. We achieve an average accuracy of 69.6 % (PMOO-DeepFP$_{\text{foi}}$) and 60.9 % (DEBORAH-DeepFP$_{\text{foi}}$), resulting in an average relative gap to PMOO-FP$_{\text{foi}}$ of only 1.17 %, and of only 2.57 % to the exhaustive DEBORAH-FP$_{\text{foi}}$ in our first dataset. When scaling to larger networks, where the existing PMOO-FP was known to struggle with computational effort, DeepFP still works. Without considerable loss of prediction accuracy we gain delay bound tightness of 1.06 % compared to standard PMOO, and 13.7 % compared to DEBORAH. In conclusion, we show that FP can considerably tighten NC delay bounds derived for FIFO multiplexing networks and that the proposed GNN-based DeepFP allows to apply it to larger networks.

APPENDIX

NETWORK CALCULUS BACKGROUND [6, 58]

A. Network Calculus System Model

NC models a network as a directed graph of connected queueing locations, the so called server graph. A server offers a resource, in communication networks forwarding of data, and a buffer to queue incoming demand, the data. Data is put into the network by flows. We assume unicast flows with a single source server and a single sink server as well as a fixed route between them. Flows' forwarding demand is characterized by functions cumulatively counting their data,

$$\mathcal{F}_0^+ = \left\{ f : \mathbb{R}^+ \to \mathbb{R}^+ \mid f(0) = 0,\ \forall s \le t\ :\ f(t) \ge f(s) \right\}. \tag{20}$$

Let functions $A(t) \in \mathcal{F}_0^+$ denote a flow's data put into a server and let $A'(t) \in \mathcal{F}_0^+$ be the flow's data put out of, both in the time interval $[0, t)$. We require the input/output relation to preserve causality by $\forall t \in \mathbb{R}^+ :\ A(t) \ge A'(t)$.

NC refines this model to one that uses bounding functions. These univariate functions (called curves) are defined independent of the start of observation, solely based on the duration of the interval of observation. By convention, let curves be in $\mathcal{F}_0$ that simply extends the definition of $\mathcal{F}_0^+$ by $\forall t \le 0 : f(t) = 0$.

Definition 1 (Arrival Curve): Given a flow with input function A, a function $\alpha \in \mathcal{F}_0$ is an arrival curve for A iff

$$\forall\, 0 \le d \le t\ :\ A(t) - A(t-d) \le \alpha(d). \tag{21}$$

Opposite to data arrivals, the forwarding service offered by some system $\mathcal{S}$ is modeled with a lower bounding curve. $\mathcal{S}$ can be a single server as above or – after applying transformations from Appendix B – a combination of multiple servers.

Definition 2 (Service Curve): If the service by system $\mathcal{S}$ for a given input A results in an output A', then $\mathcal{S}$ offers a service curve $\beta \in \mathcal{F}_0$ iff

$$\forall t : A'(t) \ge \inf_{0 \le d \le t} \{ A(t-d) + \beta(d) \}. \tag{22}$$

Definition 3 (Strict Service Curve): System $\mathcal{S}$ offers a strict service curve β to a flow if, during any busy period of duration d, the output of the flow is at least equal to $\beta(d) \in \mathcal{F}_0$.

In this paper, we restrict the set of curves to affine curves (the only type that can be used with the LUDB analysis). These curves are suitable to model token-bucket shaped data flows $\gamma_{r,b} : \mathbb{R}^+ \to \mathbb{R}^+ \mid \gamma_{r,b}(0) = 0,\ \underset{d>0}{\forall} \gamma_{r,b}(d) = b + r \cdot d,\ r, b \ge 0$, where b bounds the worst-case burstiness and r the arrival rate. Secondly, rate-latency service can be modeled by affine curves $\beta_{R,T} : \mathbb{R}^+ \to \mathbb{R}^+ \mid \beta_{R,T}(d) = \max\{0, R \cdot (d - T)\}$, $T \ge 0,\ R > 0$ where T upper bounds the service latency and R lower bounds the forwarding rate.

B. Algebraic Network Calculus Analysis

The NC analysis aims to derive a bound on the worst-case delay that a specific flow of interest (foi) experiences on its path. Service curves on that path are shared by all flows crossing the respective server yet an arrival curve is only known at the respective flow's source server. To derive the foi's end-to-end delay bound from such a model, the NC analysis relies on (min,plus)-algebraic curve manipulations.

Definition 4 (NC Operations): The (min,plus)-algebraic aggregation, convolution and deconvolution of two functions $f, g \in \mathcal{F}_0$ are defined as

$$\text{aggregation: } (f + g)(d) = f(d) + g(d), \tag{23}$$

$$\text{convolution: } (f \otimes g)(d) = \inf_{0 \le u \le d} \{ f(d-u) + g(u) \}, \tag{24}$$

$$\text{deconvolution: } (f \oslash g)(d) = \sup_{u \ge 0} \{ f(d+u) - g(u) \}. \tag{25}$$

Aggregation of arrival curves creates a single arrival curve for their multiplex. With convolution, a tandem of servers can be treated as a single system providing a single service curve. Deconvolution allows to compute an arrival curve bound on a flow's (or flow aggregate's) $A'(t)$ after crossing a system. Delay and backlog can be bounded as follows:

Theorem 3 (Performance Bounds): Consider a system $\mathcal{S}$ that offers a service curve β. Assume a flow f with arrival curve α traverses the system. Then we obtain the following performance bounds for f:

$$\text{backlog: } \forall t \in \mathbb{R}^+ :\ B(t) \le (\alpha \oslash \beta)(0) \tag{26}$$

$$\text{delay: } \forall t \in \mathbb{R}^+ :\ D(t) \le \inf \{ d \ge 0 \mid (\alpha \oslash \beta)(-d) \le 0 \} =: h(\alpha, \beta) \tag{27}$$

When bounding the residual service for a flow of interest (Theorem 1), there are some subtleties to note: the requirement on the service curve β to be strict strongly depends on the assumed multiplexing behavior. Arbitrary multiplexing needs it, FIFO does not [6]. Moreover, the arbitrary multiplexing residual service curve is not strict. In general, arbitrary multiplexing results are bounding those of any other multiplexing assumption. Compared to FIFO, we see that the residual service curves are equal for $\theta = 0$, but for any $\theta > 0$, the FIFO multiplexing can potentially give considerably more residual forwarding service.

ACKNOWLEDGMENTS The authors would like to thank the anonymous shepherd for the feedback and support.

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] P. Danielis, J. Skodzik, V. Altmann, E. B. Schweissguth, F. Golatowski, D. Timmermann, and J. Schacht, "Survey on real-time communication via Ethernet in industrial automation environments," in *Proc. of IEEE ETFA*, 2014.

[3] S. Bondorf and F. Geyer, "Generalizing network calculus analysis to derive performance guarantees for multicast flows," in *Proc. of EAI ValueTools*, 2016.

[4] A. Amari and A. Mifdaoui, "Worst-case timing analysis of ring networks with cyclic dependencies using network calculus," in *Proc. of IEEE RTCSA*, 2017.

[5] L. Thomas, J.-Y. Le Boudec, and A. Mifdaoui, "On cyclic dependencies and regulators in time-sensitive networks," in *Proc. of IEEE RTSS*, 2019.

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

[7] 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.

[8] 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.

[9] 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.

[10] M. Boyer, A. Graillat, B. Dupont de Dinechin, and J. Migge, "Bounding the delays of the MPPA network-on-chip with network calculus: Models and benchmarks," *Performance Evaluation*, vol. 143, 2020.

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

[12] P. Nikolaus and J. Schmitt, "Improving delay bounds in the stochastic network calculus by using less stochastic inequalities," in *Proc. of EAI ValueTools*, 2020.

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

[14] ——, "On the robustness of deep learning-predicted contention models for network calculus," in *Proc. of IEEE ISCC*, 2020.

[15] L. Bisti, L. Lenzini, E. Mingozzi, and G. Stea, "Estimating the worst-case delay in FIFO tandems using network calculus," in *Proc. of ICST ValueTools*, 2008.

[16] ——, "Numerical analysis of worst-case end-to-end delay bounds in FIFO tandem networks," *Real-Time Systems*, vol. 48, no. 5, pp. 527–569, 2012.

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

[18] A. Bouillard, L. Jouhet, and É. Thierry, "Service curves in network calculus: dos and don'ts," INRIA, Tech. Rep. RR-7094, 2009.

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

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

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

[22] Y. Tang, Y. Jiang, X. Jiang, and N. Guan, "Pay-burst-only-once in real-time calculus," in *Proc. of IEEE RTCSA*, 2019.

[23] M. Fidler and V. Sander, "A parameter based admission control for differentiated services networks," *Computer Networks*, vol. 44, no. 4, pp. 463–479, 2004.

[24] A. Bouillard and G. Stea, "Exact worst-case delay for FIFO-multiplexing tandems," in *Proc. of EAI ValueTools*, 2012.

[25] ——, "Exact worst-case delay in FIFO-multiplexing feed-forward networks," *IEEE/ACM Trans. Net.*, vol. 23, no. 5, pp. 1387–1400, 2015.

[26] A. Bouillard, "Trade-off between accuracy and tractability of network calculus in FIFO networks," 2020, arxiv:2010.09263.

[27] S. Bondorf and F. Geyer, "Virtual cross-flow detouring in the deterministic network calculus analysis," in *Proc. of IFIP Networking*, 2020.

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

[29] 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.

[30] 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.

[31] M. Prates, P. H. Avelar, H. Lemos, L. C. Lamb, and M. Y. Vardi, "Learning to solve NP-complete problems: A graph neural network for decision TSP," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4731–4738.

[32] F. Wang, Z. Cao, L. Tan, and H. Zong, "Survey on learning-based formal methods: Taxonomy, applications and possible future directions," *IEEE Access*, vol. 8, pp. 108 561–108 578, 2020.

[33] K. Rusek and P. Cholda, "Message-passing neural networks learn Little's law," *IEEE Commun. Lett.*, 2018.

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

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

[36] 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," vol. 38, no. 10, pp. 2260–2270, 2020.

[37] 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.

[38] T. L. Mai and N. Navet, "Deep learning to predict the feasibility of priority-based Ethernet network configurations," University of Luxembourg, Tech. Rep., 2020.

[39] F. Geyer and S. Bondorf, "Graph-based deep learning for fast and tight network calculus analyses," *IEEE Transactions on Network Science and Engineering*, 2020.

[40] R. L. Cruz, "SCED+: Efficient management of quality of service guarantees," in *Proc. of IEEE INFOCOM*, 1998.

[41] S. Bondorf and J. B. Schmitt, "Should network calculus relocate? an assessment of current algebraic and optimization-based analyses," in *Proc. of QEST*, 2016.

[42] L. Lenzini, E. Mingozzi, and G. Stea, "Delay bounds for FIFO aggregates: A case study," *Comput. Commun.*, vol. 28, no. 3,

pp. 287—299, Feb. 2005.
[43] L. Bisti, L. Lenzini, E. Mingozzi, and G. Stea, "DEBORAH: A tool for worst-case analysis of FIFO tandems," in *Proc. of ISoLA*, 2010.
[44] S. Bondorf and J. B. Schmitt, "The DiscoDNC v2 – a comprehensive tool for deterministic network calculus," in *Proc. of EAI ValueTools*, 2014.
[45] ——, "Calculating accurate end-to-end delay bounds – you better know your cross-traffic," in *Proc. of EAI ValueTools*, 2015.
[46] R. L. Cruz, "A calculus for network delay, part I: Network elements in isolation," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, pp. 114–131, 1991.
[47] B. Zhou, I. Howenstine, S. Limprapaipong, and L. Cheng, "A survey on network calculus tools for network infrastructure in real-time systems," *IEEE Access*, vol. 8, 2020.
[48] 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.
[49] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *Proc. of ICLR*, 2016.
[50] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. of ICLR*, 2018.
[51] 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.
[52] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. of NeurIPS*, 2019.
[53] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *Proc. of ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
[54] A. Guzman-Rivera, D. Batra, and P. Kohli, "Multiple choice learning: Learning to produce multiple structured outputs," in *Proc. of NIPS*, 2012.
[55] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Proc. of NIPS*, 2018.
[56] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, 2001.
[57] A. Fisher, C. Rudin, and F. Dominici, "All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously," *Journal of Machine Learning Research*, vol. 20, no. 177, pp. 1–81, 2019.
[58] A. Bouillard, M. Boyer, and E. Le Corronc, *Deterministic Network Calculus: From Theory to Practical Implementation*. John Wiley & Sons, Ltd, 2018.