End-to-end Delay Prediction Based on Traffic Matrix Sampling

Filip Krasniqi*‡, Jocelyne Elias*†, Jérémie Leguay*, Alessandro E. C. Redondi‡
*Huawei Technologies, France Research Center, France
†Dipartimento di Informatica, Scienza e Ingegneria (DISI), University of Bologna, Italy
‡Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico Di Milano, Italy

***Abstract*—In this paper we focus on the problem of predicting Quality of Service (QoS), and in particular end-to-end delay, by using traffic matrix samples. To this aim, we study different models based on machine learning as a promising tool to characterize performance in complex computer networks. More specifically, we first provide a simulation platform, based on NS 3 network simulator, in which each Origin-Destination (OD) flow is a mixture of UDP and TCP traffic and we generate useful data for our study. We present three datasets over which we gradually vary the network characteristics: incoming traffic intensity, link capacities, and propagation delays. The datasets are leveraged to train machine learning models, namely Neural Networks and Random Forests, to predict end-to-end delay starting from the knowledge of OD traffic matrix samples. The robustness of these models is evaluated in different test scenarios. Numerical results show that both models are able to accurately forecast the end-to-end delay over all tested datasets, with Random Forests outperforming Neural Networks with gaps as high as 40%.**

***Index Terms*—End-to-end delay, QoS prediction, Machine Learning, Traffic Measurement.**

I. Introduction

Since ever, computer networks have been very complex and challenging environments for both the research and industrial communities which try to understand and model their performance as well as devise efficient mechanisms for their maintenance and operation. In the last years, Software Defined Networking (SDN) has emerged as a promising paradigm to better control networks. By leveraging SDN, a (logically) centralized controller platform can fully exploit the large computing power available in the control plane and its central position to globally optimize and control networks. From now on, cutting-edge optimization and Machine Learning (ML) algorithms can be used to operate networks in real-time. This formidable opportunity transforms the way routing and traffic engineering systems should be conceived and designed.

A critical duty of the SDN controller is to determine and maintain the best appropriate routing policy so as to both optimize bandwidth usage and meet Quality of Service (QoS) requirements. In particular, the controller goal is to take routing decisions satisfying user requirements (e.g., bandwidth, low packet loss, low end-to-end delay) and network utilization constraints (e.g., link capacity constraints, maximum link load). If congestion occurs in parts of the network, the controller has to reroute flows and update forwarding rules at equipment level (e.g., switches).

Delay degradation and packet losses are generally due to congestion situations which may occur for various reasons: bursty traffic, poor configuration of buffers and queuing policies, etc. To avoid congestion, state of the art solutions are either *reactive* or *proactive*. In the reactive case, active or passive measurements of QoS metrics can be conducted (see i.e., [1]). These approaches may be costly for the operator and encounter stability issues. In the proactive case, congestion can be anticipated using analytical or machine learning models. Our paper focuses on the proactive case and especially on the application of machine learning to predict end-to-end delay based on traffic observations.

In the past, a number of analytical models to estimate the delay function (at link or end-to-end levels) have been proposed such as the Kleinrock function [2] and classic queuing models, like [3], [4], [5], [6], [7], [8] or polynomial functions [9], [10]. However, in realistic network scenarios, queuing models are either too simplistic (e.g., M/M/1 link delay model) to capture non-Poisson traffic distribution, sophisticated queuing disciplines, etc., or intractable. Therefore, in this work, we study machine learning models, which recently emerge as promising tools for learning and understanding end-to-end delay [11], [12], [13].

In particular, we provide a simulation platform to generate useful data for QoS prediction. We present three datasets over which we gradually vary the characteristics of the environment (including link capacities and propagation delays). We show using Random Forests and Neural Networks that end-to-end delay can be predicted based on traffic matrix samples and we evaluate the robustness of these models against missing inputs (e.g., knowledge of link capacities) and perturbations of the environment (e.g., randomly evolving propagation delays). From numerical results, we observe that Random Forests are able to predict accurately the end-to-end delay over all tested datasets, outperforming Neural Networks with performance gaps as high as 40%.

Differently from what has been done in [11], [12], [13], we use the NS 3 network simulator to conduct extensive simulations with accurate traffic patterns for UDP and TCP, and different link properties (capacities and propagation delays). We measure not only the end-to-end delay but also the per-link throughput, delay and packet losses. However, for our contribution on QoS prediction, we focus on the end-to-end

delay. To summarize, our main contributions are fourfold: 1) a simulation platform based on NS 3 to produce realistic data, 2) several usable datasets for QoS prediction, 3) machine learning models for the prediction of end-to-end delay, and 4) a performance evaluation of these models.

The paper is structured as follows. Section III-A states our problem, while Section II discusses related works. Section III gives information on the generated data sets. Finally, Section IV illustrates and analyzes numerical results that show the efficiency of ML techniques in learning the delay and Section V concludes the paper.

II. Related Work

In this section, we discuss the most relevant work [11], [12], as well as RouteNet [13] related to learning and inferring the QoS performance metrics in a network.

In [11], the authors adopt a fully-connected feed-forward neural network to model the mean delay of a set of networks using as input the traffic matrix. The main goal is to understand how fundamental network characteristics (such as traffic intensity) relate with basic neural network parameters (e.g.: shallowness of the neural network). The authors used the Omnet++ simulator (version 4.6) and considered different, yet simple, network topologies (unidirectional ring, star and scale-free networks) and traffic parameters (different packet lengths and traffic intensities) in order to evaluate the impact of these parameters on the learning model. They provided several insightful conclusions which can be summarized as follows: 1) Simple NNs are affected by the additional complexity introduced in the different scenarios. This suggests that the delay function is complex and multi-dimensional, requiring sophisticated regression techniques such as deep NNs. 2) For the considered scenarios, the topology and the routing configuration have no impact on the accuracy.

Deep-Q [12] is a data driven system that aims at modeling the distribution of the QoS of a network given traffic conditions, using a Deep Generative Network (DGN). More specifically, the authors of Deep-Q design a QoS inference structure with a Long Short-Term Memory (LSTM)-enhanced Variational Auto-Encoder (VAE). The LSTM module is used to extract fine-grained traffic features and long-term dependencies while the VAE module learns to reconstruct the QoS distribution from the traffic features.

Some authors of ref. [11] with other researchers proposed later RouteNet [13], which is based on a Graph Neural Networks (GNN) model and able to produce quite accurate estimates of performance metrics - delay and jitter - in more general scenarios: in fact, in contrast to [11], [12], it does not assume a fixed topology and/or a routing scheme, and thus it aims at working with arbitrary topologies and routing schemes not seen during the training phase.

Differently from previous work, we first provide an open simulation environment to produce new datasets with a clear description of traffic generation and measurement models. We consider a realistic mixture of different traffic types (i.e., UDP and TCP) as it impacts QoS, and we measure a larger set of indicators including the per link delay/load/packet loss, both at link and end-to-end levels. In Deep-Q [12], the authors used real traces captured on their testbeds, however, they do not give details about these traces. Furthermore, they present their results focusing on a very simple topology (a single hop link) or considering specific paths (the most congested one) in a quite small network. With regards to end-to-end delay prediction, we analyze the robustness of Random Forests and Neural Networks models against different network configurations (i.e., link capacities) and uncontrollable characteristics of the environment (i.e., time-varying propagation delays).

III. Methodology and Datasets

As argued previously, the network is a complex environment, where it can be hard to predict QoS performance using standard methods such as queuing models. Therefore, we study the use of machine learning to learn and estimate end-to-end delay. This section explains how we generated our own datasets, and it details their characteristics.

A. Methodology

The end-to-end delay is impacted by the tight relationship between the traffic demand, the network topology and the network performance [14], that are in turn induced by a number of factors. For instance, the traffic demand depends on the traffic intensity, the nature of the random process generating packets or the mixture of application types for all source and destination pairs. The network topology is itself characterized by parameters such as the graph connectivity, link capacities or queue sizes. And also, network performance can be influenced by propagation delays, queuing disciplines or routing policies. In the general case, we can clearly see that we need to learn a complex function f from inputs like traffic measurements (M), topology parameters (T), routing policies (R), etc., to predict an end-to-end delay vector $D_{\mathcal{F}}$ for a set of flows:

$$D_{\mathcal{F}} = f(M, T, R, ...).$$

As the number of influencing factors is too large in practice, our long term objective is to define the simplest model as possible and identify the subset of relevant input data (i.e., features) to achieve good accuracy and good robustness in different environments.

To make a first step in this direction, we study the case where the network topology and the routing policy ($R =$ Shortest Path First algorithm) are fixed, but we vary three environmental characteristics that have a strong impact on the delay: i) traffic intensity, ii) link capacities, and iii) link propagation delays. In particular, we will consider three scenarios with increasing degree of freedom / complexity:

- **Traffic Only (TO)**: fixed capacity and propagation delay;
- **Traffic and Capacity (TnC)**: fixed propagation delay and variable capacity;
- **Traffic and Capacity with Delays (TnCwD)**: all three parameters (traffic intensity, link capacity and link propagation delay) are variable.

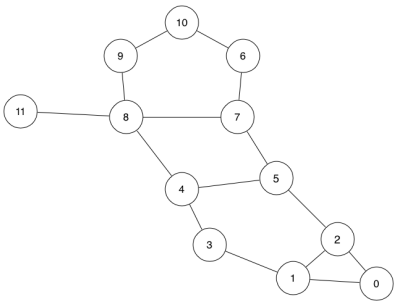


Fig. 1: The Abilene network topology.

In Section IV we will study how machine learning models perform when taking as input traffic matrix samples. We will consider whether extra knowledge about the environment, such as link capacities, improves accuracy.

B. Network simulations

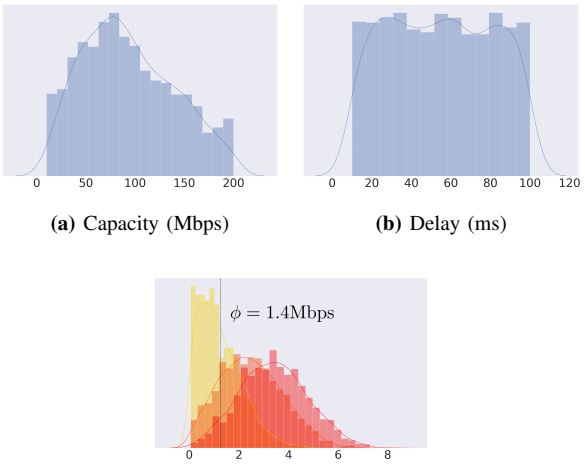
To generate realistic datasets, we implemented a framework based on the NS 3 network simulator [15]. This framework permits to generate and measure traffic with specific characteristics over a pre-defined network topology.

We considered the well-known Abilene [16] network (12 nodes and 15 links), illustrated in Figure 1. Link capacities C and link propagation delays D are generated randomly according to a truncated Gaussian distribution $\mathcal{C}$ having ranges in [10 Mbps, 200Mbps] and a uniform distribution $\mathcal{D}$ within [10ms, 100ms], respectively. Figure 2 shows the empirical distributions observed in the instances we generated for link capacities, link delays and OD flow rates.

In each simulation, nodes in the network are instructed to generate traffic towards all other nodes. Each OD flow is a mixture of two traffic types: (i) a constant bit rate (CBR) UDP flow at a low bitrate r = 128Kbps, used as background traffic, and (ii) either a CBR UDP flow or a TCP flow (uniformly chosen at random). The application rate R of this latter TCP or UDP flow is drawn from a Gaussian distribution with mean r_k and variance 1.5Mbps, truncated within the range [50Kbps, 40Mbps]. The specific value of r_k controls the level of traffic intensity, and is chosen according to the link capacities C drawn during the generation of the network instance, avoiding the creation of traffic flows that would likely result in excessive congestion. In detail, let ϕ be the bottleneck link capacity normalized by the maximum number of flows passing through it: we define ten different distributions of traffic intensity $\mathcal{I}_k$ (where $k = 1$ refers to the lowest intensity and $k = 10$ to the highest), with mean r_k chosen according to the following:

$$r_k = \phi + (k \times 7 - 20) \times 50Kbps. \quad (1)$$

Figure 2c shows three of such distributions, corresponding to $k = 1$, 7 and 10 for a value of $\phi = 1.4$Mbps. Summarizing, each single simulation run s is determined by a specific drawing of link capacities $C \sim \mathcal{C}$, propagation delays $D \sim \mathcal{D}$ and traffic intensities $I \sim \mathcal{I}_k$, according to the selected intensity level k.



(a) Capacity (Mbps) **(b)** Delay (ms)

(c) Traffic (Mbps)

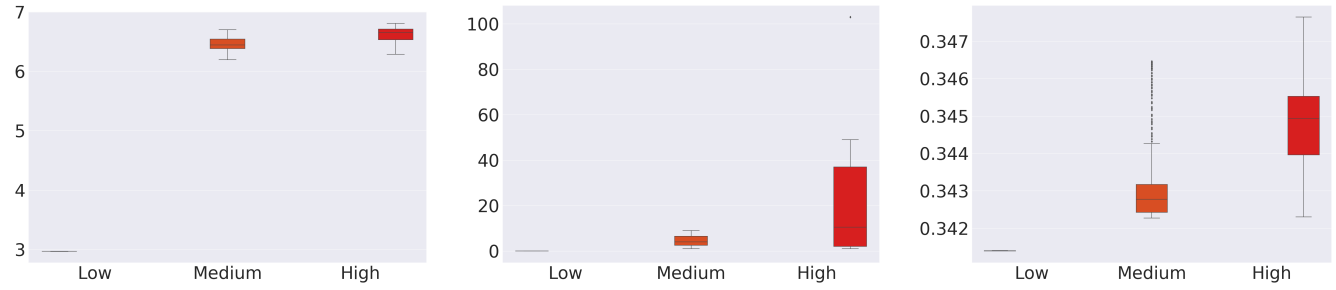
Fig. 2: Distribution, obtained from sampling corresponding real distribution, of capacity $\mathcal{C}$ (a), propagation delay $\mathcal{D}$ (b) and traffic intensity $\mathcal{I}_k$ (c), for low intensity ($k = 1$, yellow), medium intensity ($k = 7$, orange) and high intensity ($k = 10$, red).

C. Dataset generation

Each single simulation $s = (C, D, I)$ runs for 50 seconds of NS 3 time[1] and we measure the corresponding OD traffic matrix every $T = 0.1$s, for a total of 500 traffic matrix samples per simulation (12-by-12). Traffic matrix measurements are performed using NS 3 callbacks, which allow to track the amount of bits output by each node towards each other destination at specific time intervals. The first 10 seconds of measurements (i.e., the initial 100 traffic matrices) are discarded in order to neglect any transient traffic regime (e.g., TCP slow start). The remaining 400 matrices are averaged over with a rolling window of length $T_w = 5$s (i.e., each period contains 50 matrices) and moving with a step size of 0.1s, resulting in a total of 350 observation periods. Within each period, the mean and standard deviation of each OD flow are computed, resulting in a total of $2 \times 12 \times 11 = 264$ features. Concurrently, in each period the average end-to-end delay matrix is extracted from the simulation and stored as ground truth (for a total of $12 \times 11 = 132$ end-to-end delay values per period). To compute such a delay, we leverage NS 3 timer functionalities by tracking the reception instant and the transmission instant of each IP packet for each OD flow, and taking the difference. In addition to means and standard deviations, we could have also used quantiles to characterize traffic, similarly to what has been done for QoE prediction [17]. However, to keep the number of features manageable, we stick to two features per OD flow and leave more advanced input data for future work.

As mentioned before, we considered in this work three

[1]The actual time spent for one simulation is much higher, varying between 30 minutes and 3 hours (depending on the traffic intensity) on a high-end simulation machine, due to the large amount of packets to process.



(a) Load (Mbps) **(b)** Dropped Packets (%) **(c)** Delay (ms)

Fig. 3: Distribution of load (a) and dropped packets (b) for different traffic intensities on link 4-8. Distribution of the end-to-end delay (c) for the longest OD flow crossing link 4-8 for different traffic intensities.

different datasets of increasing complexity, corresponding to the three following scenarios:

1) Traffic Only (TO): we run different simulations with the same values of link capacities C and propagation delays D, changing only the traffic intensities I. For each intensity level k from 1 to 10 we ran 200 simulations, for a total of 2000 simulations and $2000 \times 350 = 700$K observations.
2) Traffic and Capacity (TnC): simulations are all run with the same values of propagation delays D, but with different traffic intensities I and link capacities C. In details we ran simulations with 50 different realizations of link capacities and 4 different realizations for each one of the 10 traffic intensity levels, for a total of $50 \times 10 \times 4 = 2000$ different simulations and again 700K observations.
3) Traffic and Capacity with Delays (TnCwD): finally, the most challenging scenario consists in varying all parameters. This time we generate 50 different realizations of link capacities C, 10 realizations of propagation delays D and one realization for each one of the 10 traffic intensities, for a total of $50 \times 10 \times 10 = 5000$ simulations, corresponding to 1.75M observations.

D. Dataset description

Let us have a closer look at the network instances we generated. Figure 3a shows the distribution of the load observed on the link 4-8, which is the one with the highest probability of being the bottleneck link in the Abilene topology given the routing obtained executing Dijkstra's algorithm, for three different values of traffic intensity level k (1,7 and 10), and when the link capacity is equal to 70 Mbps. As one can see, the observed load increases as the traffic intensity level increases, saturating the link capacity. We also show in Figure 3b the distributions of dropped packets on link 4-8, corresponding to the same three traffic intensity levels. As expected, when increasing the traffic intensity (and therefore the link utilization), the distribution of the dropped packets moves towards bigger values and widens. Finally, Figure 3c shows how the end-to-end delay distribution changes when varying only the intensity of the traffic (i.e., link capacities and propagation delays are fixed). We focused on the OD flow associated to the longest path resulting from the routing algorithm (flow 0-9, 5 hops including the link 4-8). As one can see, higher traffic intensity values are associated to higher end-to-end delay values with increasing variance.

IV. Applying Machine Learning Models

In this section, we introduce the machine learning models we used to predict end-to-end delay and we present numerical results based on simulations.

A. Machine Learning Models

The datasets generated according to the procedure described in the previous section are used to train two machine learning models, namely Random Forests (RF) and Neural Networks (NN), both well known for their capacity of capturing complex, non linear relationships between inputs and targets. For each considered scenario, the datasets have been split in training and test sets according to a 80%-20% ratio, and according to the following steps:

- *TO*: the dataset contains 200 realizations for each traffic intensity k. We take 160 realizations for training and 40 for testing for each traffic intensity.
- *TnC*: the training set is created by sampling 40 out of the 50 realizations of link capacities, and retaining for each one all the 4 realizations for each traffic intensity.
- *TnCwD*: in this case we select 8 of the 10 different realizations of propagation delay, each corresponding to 50 different realizations of links capacity and 10 different intensity levels.

Both RF and NN require several input hyper-parameters, whose setup is not trivial and need to be optimized. Therefore, for each scenario, the training set is further split in a sub-training set and a validation set, according to 5-fold cross-validation with a ratio of 80-20. At each iteration of the cross-validation phase, the sub-training set is trained with a fixed set of hyper-parameters, and performance are computed on the test set. The performance measure used is the MSE between the ground truth and predicted end-to-end delay. Finally, the

Dataset	TO	TnC	TnCwD
Neural Network (NN)	Learning Rate: 0.003 λ: 0.0001 Num.Layers: 12 Hidden Nodes: 264	Learning Rate: 0.005 λ: 0.001 Num.Layers: 12 Hidden Nodes: 264	Learning Rate: 0.005 λ: 0.005 Num.Layers: 20 Hidden Nodes: 264
Random Forest (RF)	Max Depth: 58 Max Features: 79 Num. estimators: 59	Max Depth: 60 Max Features: 153 Num. estimators: 63	Max Depth: 58 Max Features: 100 Num. estimators: 63

TABLE I: Best hyper-parameters found in all scenarios.

hyper-parameters corresponding to the best performance are selected. For what concerns RF we used the Python scikit-learn implementation and the following space of hyper-parameters have been explored through Grid search: number of estimators between 32 and 64, max depth between 32 and 48, max number of features to consider when looking for the best split between 64 and 192. As for NN we used the PyTorch framework and we considered the following ranges for the hyper-parameters: number of hidden layers in [1-20], number of neurons per layer in [192 - 264] (we use the empirical rule of having a number of neurons approximately equal to the number of input features or equal to the square root of the product between input features and output targets), learning rate in [10^{-3}, 5×10^{-3}] and regularization rate λ in [10^{-4}, 5×10^{-3}]. The neural network uses ReLU as activation function, Adam as optimizer and a batch size of 512 observations.

Table I shows the finally selected parameters for RF and NN models. The following observations can be made:

- For what concerns RF, the hyper-parameters search tends to provide a complex model (high number of estimators and max depth). These parameters affect both the time spent for the training phase (6 hours on average) and the size required to store the trained model (as large as 10GB in the configuration of hyper-parameters requiring the largest amount of memory).
- As for NN, the best architecture found is composed of 12 layers for TO and TnC scenarios, and 20 layers for TnCwD, with 264 neurons per layer and specific learning rate and regularization parameter λ. The corresponding computational and memory resources needed to train and store the network are about 30 minutes and 4 MBs, respectively, therefore greatly outperforming RF.

B. Numerical results

We report here the prediction results obtained on the test sets for the three considered scenarios. We recall that for each scenario the machine learning models were trained with samples spanning the entire set of traffic intensity distributions (from low to high). For what concerns the test phase, we show prediction results grouped by intensity level k, computing the Root Mean Squared Error (RMSE) restricted to those test samples corresponding to a particular intensity level. This allows to better understand the impact of increasing traffic intensity on the performance prediction. To provide a general result we provide in Figure 4 boxplots of the RMSE distribution for all tested scenarios, three representative traffic intensities (low, medium and high, corresponding to $k = 1$, 7 and 10) and the two tested machine learning models. Each boxplot is computed as follows: first, we take the average RMSE over all end-to-end delay predictions of a single simulation (132×350). Then, we compute the distribution of such average RMSE over all simulations in the test set for each specific scenario (i.e., $0.2 \times 200 = 40$ values for TO and TnC, and $0.2 \times 500 = 100$ values for TnCwD). Blueish boxplots in Figure 4 correspond to RF and reddish ones to NN, with color intensity related to traffic intensity (from low to high).

Furthermore, we complete our experiments with the following tests:

- *Average value predictor*: we always compare the obtained performance with the one of a baseline predictor, which always outputs the average end-to-end delay over all the training samples. Such performance is illustrated using the green dashed lines in Figure 4.
- *Knowledge of link capacities*: for those scenarios where link capacities are variant (i.e., TnC and TnCwD), we also consider a different version of the machine learning models, where the specific configuration of link capacities C is also passed as input during training (elevating the number of input features from 264 to 279).

We start commenting on the performance of the machine learning models trained without link capacities. Comparing Figure 4(a) with Figure 4(c), the following observations can be made:

1) In general, and as expected, the average and variance of the RMSE increase as both the traffic intensity and the scenario complexity increase. Both RF and NN show outstanding performance for the first two scenarios (TO and TnC), while in the TnCwD scenario the error is considerably higher. RF improves the baseline predictor by 84%, 80% and 50%, on average, in the TO, TnC and TnCwD scenarios, respectively. As for NN, the improvement is limited to 77%, 73% and 37%.
2) We observe that RF have better performance than NN in all the datasets. The average gap is as high as 40% for TO, 27% for TnC and 18% for TnCwD. We observe that the gap between RF and NN decreases as the scenario complexity increases: this is particularly interesting, also considering the reduced time and memory resources required by NN compared to RF.

For the performance obtained when the models are trained also with link capacities as inputs, results are shown in Figure 4(b) and Figure 4(d). We observe that for RF, there is no visible difference in the average error between passing the link capacities in input or not (differences are within 1%-3%). For what concerns NN, we measure an improvement in the average RMSE (about 8%) only for the TnCwd scenario, together with a reduction of the error variance. Such limited impact of passing link capacities as input of the models during training can be explained considering that (i) link capacity features are few compared to the traffic matrix samples (15 vs 264), and especially for RF they have a smaller chance to be selected

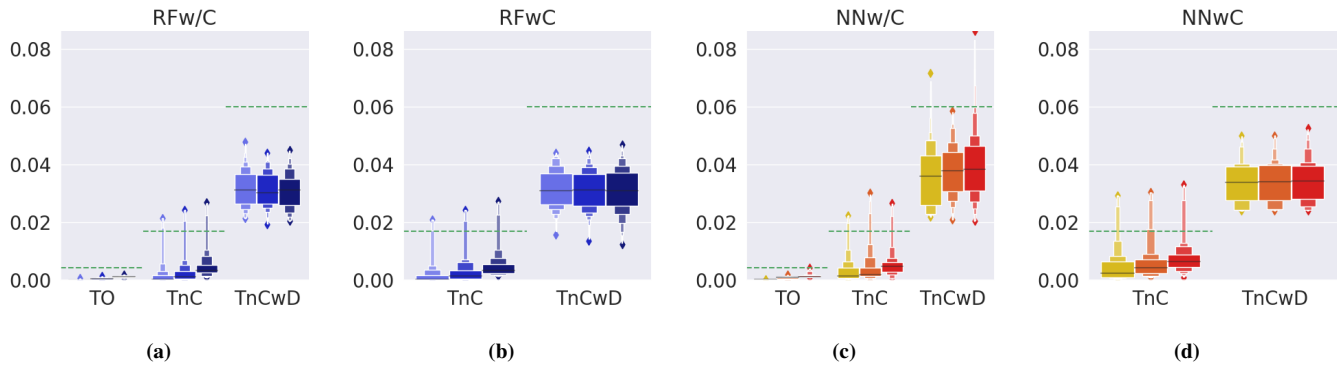


Fig. 4: Distribution of the end-to-end delay prediction RMSE in the different tested scenarios: (a) and (b) refer to random forest without (**w/C**) and with (**wC**) link capacities as extra features, while (c) and (d) refer to neural network without (**w/C**) and with (**wC**) capacities as extra features. Increasing color intensity corresponds to increasing traffic intensity (*low* with $k = 1$, *medium* with $k = 7$, *high* with $k = 10$). Green lines correspond to the average value baseline predictor.

by each estimator, (ii) the number of different realizations of link capacities is quite limited (only 50) and thus may not be enough to completely characterize the relationship with end-to-end delay for different traffic intensities, and (iii) the traffic matrix samples have already a high correlation with the end-to-end delay in our data as nearly 50% of traffic is TCP.

V. Conclusions

In this paper, we considered the problem of efficiently predicting QoS metrics in a computer network, focusing in particular on the end-to-end delay. We studied two machine learning models, Neural Networks and Random Forests, which are particularly adapted as we demonstrated in our simulation campaign. We first provided a simulation platform (based on NS 3), considering a realistic mix of UDP and TCP traffic for origin-destination flows, which allowed us to generate meaningful data for QoS (end-to-end delay) prediction in a network. We further produced three datasets over which we gradually varied the key network's features that impact delay: incoming traffic intensity, link capacities, and propagation delays. Finally, with a thorough numerical analysis we demonstrated that the end-to-end delay can be effectively predicted by using traffic matrix samples, and we further evaluated and quantified the robustness of our proposed models against missing inputs and some perturbations in the network.

Finally, numerical results show that both models are able to predict accurately the end-to-end delay over all tested datasets, with Random Forests outperforming Neural Networks and a baseline predictor with gaps as high as 40% and 84%, respectively.

Future extensions of this work that we deem promising include the utilization of graph neural networks, able to leverage the specific topology of the network, and considering routing as input feature so that our ML models could generalize and be used for QoS routing. To allow for reproducible research, the NS 3 framework used in this paper and the datasets are publicly available online at https://filipkrasniqi.github.io/QoSMLpresentation/.

References

[1] Olivier Tilmans, Tobias Bühler, Ingmar Poese, Stefano Vissicchio, and Laurent Vanbever. Stroboscope: Declarative network monitoring on a budget. In *Proc. USENIX NSDI*, 2018.

[2] Leonard Kleinrock. *Communication nets: Stochastic message flow and delay*. Courier Corporation, 2007.

[3] Bernard Fortz and Mikkel Thorup. Internet traffic engineering by optimizing ospf weights. In *Proc. IEEE INFOCOM*, 2000.

[4] Jennifer Rexford. Route optimization in IP networks. *Handbook of Optimization in Telecommunications, Springer*, pages 679–700, Boston, MA, 2006.

[5] Walid Ben-Ameur and Adam Ouorou. Mathematical models of the delay constrained routing problem. *Algorithmic Operations Research*, 1(2), 2006.

[6] Jérôme Truffot, Christophe Duhamel, and Philippe Mahey. k-Splittable delay constrained routing problem: A branch-and-price approach. *Networks*, 55(1):33–45, 2010.

[7] Bernard Fortz, Luis Gouveia, and Martim Joyce-Moniz. Models for the piecewise linear unsplittable multicommodity flow problems. *European Journal of Operational Research*, 261(1):30–42, 16 August 2017.

[8] Racha Gouareb, Vasilis Friderikos, and A Hamid Aghvami. Delay sensitive virtual network function placement and routing. In *Proc. IEEE ICT*, 2018.

[9] Ariel Orda, Raphael Rom, and Nahum Shimkin. Competitive routing in multi-user environments. *IEEE/ACM Transactions on Networking*, 1(5):510–521, October 1993.

[10] Eitan Altman, Tamer Basar, Tania Jimenez, and Nahum Shimkin. Competitive routing in networks with polynomial costs. *IEEE Transactions on automatic control*, 47(1):92–96, 2002.

[11] Albert Mestres, Eduard Alarcon, Yusheng Ji, and Albert Cabellos-Aparicio. Understanding the modeling of computer network delays using neural networks. In *Proc. ACM BigDaMa Workshop*, 2018.

[12] Shihan Xiao, Dongdong He, and Zhibo Gong. Deep-Q: Traffic-driven QoS Inference using Deep Generative Network. In *Proc. ACM NetAI Workshop*, 2018.

[13] Krzysztof Rusek, Jos Suarez-Varela, Albert Mestres, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Unveiling the potential of graph neural networks for network modeling and optimization in sdn. In *arXiv preprint arXiv:1901.08113*, 2019.

[14] Thomas Bonald and James W Roberts. Internet and the erlang formula. *ACM SIGCOMM Computer Communication Review*, 42(1):23–30, 2012.

[15] George F. Riley and Thomas R. Henderson. *The ns-3 Network Simulator*, pages 15–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[16] Anukool Lakhina, Konstantina Papagiannaki, Mark Crovella, Christophe Diot, Eric D Kolaczyk, and Nina Taft. Structural analysis of network traffic flows. In *ACM SIGMETRICS Performance eval. review*, 2004.

[17] Michael Seufert, Pedro Casas, Nikolas Wehner, Li Gang, and Kuang Li. Features that matter: Feature selection for on-line stalling prediction in encrypted video streaming. In *Proc. IEEE INFOCOM Workshops*, 2019.