Low Complexity Adaptive ML Approaches for End-to-End Latency Prediction

Pierre Larrenie, Jean-François Bercher, Olivier Venard, Iyad Lahsen-Cherif

LOW COMPLEXITY ADAPTIVE MACHINE LEARNING APPROACHES FOR END-TO-END LATENCY PREDICTION *

Pierre Larrenie
Thales SIX & LIGM
Université Gustave Eiffel, CNRS
Marne-la-Vallée, France
pierre.larrenie@esiee.fr

Jean-François Bercher
LIGM
Université Gustave Eiffel, CNRS
Marne-la-Vallée, France
jean-francois.bercher@esiee.fr

Olivier Venard
ESYCOM
Université Gustave Eiffel, CNRS
Marne-la-Vallée, France
olivier.venard@esiee.fr

Iyad Lahsen-Cherif
Institut National des Postes et Télécommunications (INPT)
Rabat, Morocco
lahsencherif@inpt.ac.ma

ABSTRACT

Software Defined Networks have opened the door to statistical and AI-based techniques to improve efficiency of networking. Especially to ensure a certain *Quality of Service* (QoS) for specific applications by routing packets with awareness on content nature (VoIP, video, files, etc.) and its needs (latency, bandwidth, etc.) to use efficiently resources of a network.

Monitoring and predicting various Key Performance Indicators (KPIs) at any level may handle such problems while preserving network bandwidth.

The question addressed in this work is the design of efficient, low-cost adaptive algorithms for KPI estimation, monitoring and prediction. We focus on end-to-end latency prediction, for which we illustrate our approaches and results on data obtained from a public generator provided after the recent international challenge on GNN [12].

In this paper, we improve our previously proposed low-cost estimators [6] by adding the adaptive dimension, and show that the performances are minimally modified while gaining the ability to track varying networks.

Keywords KPI Prediction · Machine Learning · Adaptivity · General Regression · SDN · Networking

1 Introduction

Routing while ensuring quality of service (QoS) remains a significant challenge in all networks. Whatever the resources, their use must be optimized to satisfy both throughput and QoS to users. This is true for static wide area networks, but even more so for mobile networks with dynamic topology.

The emergence of software-defined networks (SDNs) [11, 1] has enabled data to be shared more efficiently across communication layers. Services can provide network requirements to routers; routers acquire data about network performance and allocate resources to meet those requirements as best as possible. However, acquiring overall network performance can result in high network bandwidth consumption for signaling, degrading the available resources, and is particularly limiting for resource-constrained networks such as mobile networks (MANETs).

We consider a network for which we wish to reduce signaling and perform intelligent routing. In order to limit the amount of signaling, the first axis is to estimate some key performance indicators (KPIs) from other KPIs. A second axis would be to perform this prediction locally, at the node level, rather than a global estimation in the network. Finally, if predictions are to be performed locally, the complexity of the algorithms will need to be low while preserving good prediction quality. The last point is to be able to detect and track changes in the state of the network, which implies that the predictors will have to use only a small number of the previous states of the network and be able to readapt continuously.

*__Note__: Paper accepted at the 5th International Conference on Machine Learning for Networking (MLN'2022) and will be published as a post-proceedings in Springer's LNCS.

The question addressed in this work is the design of efficient, low-cost adaptive algorithms for KPI estimation, monitoring and prediction. In the present paper, we improve our previously proposed low-cost estimators [6] by adding the adaptive dimension and show that the performances are minimally modified while gaining the ability to track varying networks. We focus on end-to-end latency prediction, for which we illustrate our approaches and results on data we generated using a public generator made available after the recent international challenge [12]. The best performances of the state-of-the-art are obtained with Graph Neural Networks (GNNs) [10, 3, 12]. Although this is a global method while we favor local and adaptive methods, we used these performances as a benchmark.

We present related works in section 2. Then we present in section 3 our main results from [6]. Instead of using high performances global and high-costs methods based on Graph Neural Networks (GNNs) [10, 3, 12], we proposed to use standard machine learning regression methods. We showed that a careful feature engineering and feature selection (based on queue theory and the approach in [2]), as well as the use of a single feature with curve-fitting methods, allows to obtain near state-of-the-art performances with both a very low number of parameters, significantly lower learning and inference times compared to GNNs, and the with the ability to operate at the link level instead of a whole-graph level. In section 4, we show how these block algorithms can be transformed into versions implementable in an iterative way (i.e. by taking into account the data one by one as they become available), with the originality of using a regularization term. Then, time dependent estimations, or the addition of forgetting factor will give them an adaptive character. In section 5 we describe the validation dataset we built from a public generator and then the results of our experimentation. Finally, we conclude, discuss the overall results and draw some perspectives.

2 Related work

[4] present an heuristic and an Mixed Integer Programming approach to optimize Service Functions Chain provisioning when using Network Functions Virtualization for a service provider. Their approach relies on minimizing a trade-off between the expected latency and infrastructures resources.

Such optimization routing flow in SDN may need additional information to be exchanged between the nodes of a network. This results in an increase of the volume of signalization, by performing some measurements such as in [7]. This is not a consequent problem in unconstrained networks, i.e. static wired networks with near-infinite bandwidth but may decrease performance of wireless network with poor capacity. An interesting solution to save bandwidth would be to predict some of the KPIs from other KPIs and data exchanged globally between nodes.

In [8, 9], authors proposed a MANETs application of SDN in the domain of tactical networks. They proposed a multi-level SDN controllers architecture to build both secure and resilient networking. While orchestrating communication efficiently under military constraints such as: high-level of dynamism, frequent network failures, resources-limited devices. The proposed architecture is a trade-off between traditional centralized architecture of SDN and a decentralized architecture to meet dynamic in-network constraints.

[5] proposed a Quality of Experience (QoE) management strategy in a SDN to optimize the loading time of all the tile of a mapping application. They have shown the impact of several KPIs on their application using a Generalized Linear Model (GLM). This mechanism make the application aware of the current network state.

[10] used GNNs for predicting KPIs such as latency, error-rate and jitter. They relied on the *Routenet* architecture of Figure 1. The idea is to model the problem as a bipartite hypergraph mapping flows to links as depicted on Figure 2. Aggregating messages in such graph may result in predicting KPIs of the network in input. The model needs to know the routing scheme, traffic and links properties. Their result is very promising and has been the subject of two ITU Challenge in 2020 and 2021 [3, 12]. These ITU challenges have very good results since the top-3 teams are around 2% error in delay prediction in the sense of Mean-Absolute Percentage Error (MAPE).

In [2], very promising results were obtained with a a near 1% GNN model error (in the sense of MAPE) on the test set. The model mix analytical $M/M/1/K$ queueing theory used to create extra-features to feed GNN model. In order to satisfy the constraint of scalability proposed by the challenge, the first part of model operates at the link level.

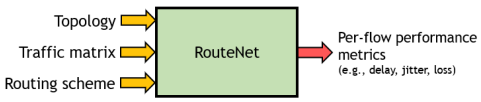
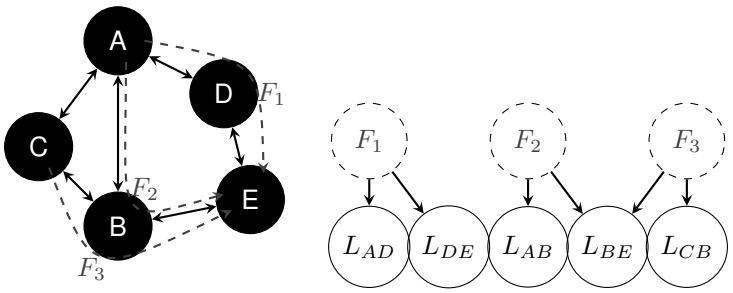


Figure 1: Routenet Architecture [10]



(a) Simple topology (b) Paths-links Hypergraph of (a)

Figure 2: Routenet [10] paths-links hypergraph transformation applied on a simple topology graph carrying 3 flows.

(a) Black circles represents communication node, double headed arrows between them denotes available symmetric communications links and dotted arrows shows flows path. (b) Circle (resp. dotted) represents links (resp. flows) entities defined in the first graph (L_{ij} is the symmetric link between node i and node j.). Unidirectional arrows encode the relation "<flow> is carried by <link>".

3 Simple machine-learning approaches for latency prediction

Our first problem is to define an estimator $\hat{y}$ of the occupancy y as a function of the different available "features" of the system, with a joint objective of low complexity and performance. To do so, we will look for an approximation function $f_{\boldsymbol{\theta}}(\mathbf{u})$ that allows to estimate y from the features $\mathbf{u}$ and parameters $\boldsymbol{\theta}$.

$$\hat{y} = f_{\boldsymbol{\theta}}(\mathbf{u}) \tag{1}$$

Here $\mathbf{u}$ and $\boldsymbol{\theta}$ are vectors that collect the different features or parameters. Once an estimate of occupancy is obtained, it is possible to get the latency prediction $\hat{d_n}$ for a specific link n by the simple relation

$$\hat{d_n} = \hat{y_n} \frac{\mathbb{E}(|P_n|)}{c_n} \tag{2}$$

where $\mathbb{E}\left[|P_n|\right]$ is the observed average packet size on link n and c_n the capacity of this link.

For analytical simplicity, the parameters $\boldsymbol{\theta}$ will be sought by minimizing the minimum mean square error

$$\mathbb{E}\left[(y - \hat{y})^2\right] = \mathbb{E}\left[(y - f_{\boldsymbol{\theta}}(\mathbf{u}))^2\right], \tag{3}$$

although the performances are also often evaluated in the MAPE sense

$$\mathcal{L}\left(\hat{y}, y\right) = \frac{100\%}{N} \sum_{n=1}^{N} \left| \frac{\hat{y}_n - y_n}{y_n} \right| \tag{4}$$

which is preferred to Mean Squared Error (MSE) because of its scale-invariant property.

We will focus here on two very simple models, although other machine learning models have also been considered in [6]. Indeed, these two models lend themselves very easily to an adaptive formulation. In this section, we will first describe these two approaches and their performances, before giving the general adaptive formulation, which we will particularize in both cases.

3.1 Feature Engineering and Linear Regression

Based on the assumption that the system may be approximated by a model whose essential features come from $M/M/1/K$ and $M/G/1/K$ queue theory, we took essential parameters characterizing queueing systems, such as: ρ, ρ_e, π_0, π_K, etc. and built further features by applying interactions and various non-linearities (powers, log, exponential, square root). Then, we selected features in this set by a forward step-wise selection method; i.e. by adding in turn each feature to potential models and keeping the feature with best performance. Finally, we selected the model with best MAPE error. For a linear regression model, this led us to select and keep a set of 4 simple features, which interestingly enough, have simple interpretations:

$$\begin{cases} \pi_0 = \frac{1-\rho}{1-\rho^{K+1}} \\ L = \rho + \pi_0 \sum_k k\rho^k \\ \rho_e = \frac{\lambda_e}{\lambda}\rho = \frac{\lambda_e}{\mu} \\ S_e = \sum_k k\rho_e^k \end{cases} \tag{5}$$

where L is the expected number of packets in the queue according to $M/M/1/K$, π_0 the probability that the queue is empty according to $M/M/1/K$ theory, ρ_e the effective queue utilization, and S_e the

unnormalized expected value of the effective number of packet in the queue buffer. These features can be thought as a kind of data preprocessing, before applying ML algorithms, and this turns out to be a key to achieving good performances. The 4 previous features have been used as input for several machine learning models like Multi-Layer Perceptron model (MLP), Linear Regression, SVM, Random Forest, Gradient Boosting Regression Tree. We only describe here the case of linear regression, since it is a method for which an adaptive version is readily obtained. In this case, model (1) is simply

$$\hat{y} = \theta_0 + \theta_1\pi_0 + \theta_2 L + \theta_3\rho_e + \theta_4 S_e = \boldsymbol{\theta}^T\mathbf{u} \tag{6}$$

with $\boldsymbol{\theta}^T = [\theta_0, \dots \theta_4]$ and $\mathbf{u}^T = [1, \pi_0, L, \rho_e, S_e]$. For the linear regression model in ((6), it is well known that the regularized minimum mean squared error

$$J(\boldsymbol{\theta}) = \mathbb{E}\left[(y - \boldsymbol{\theta}^T\mathbf{u})^2\right] + \alpha||\boldsymbol{\theta}||^2 \tag{7}$$

is obtained for

$$\boldsymbol{\theta} : (\mathbf{R}_{uu} + \alpha\mathbf{1})\,\boldsymbol{\theta} = \mathbf{R}_{yu} \tag{8}$$

where we denoted

$$\begin{cases} \mathbf{R}_{uu} = \mathbb{E}\left[\mathbf{u}\mathbf{u}^T\right], & \text{the correlation matrix of } \mathbf{u} \\ \mathbf{R}_{yu} = \mathbb{E}\left[y\mathbf{u}\right], & \text{the correlation vector of } y \text{ and } \mathbf{u} \end{cases}$$

and $\mathbf{1}$ the identity matrix, α the regularization parameter.

As far as performance is concerned with this approach, it was evaluated using static data from the GNN ITU Challenge 2021 [12]. Compared to the state-of-the-art, our linear regression with carefully selected features shows a very slight performance degradation: 1.74% in MAPE while the best state-of-the-art method is at 1.27%. One strong advantage is in term of training and inference time. It has a training time of less than a second when GNN requires more than 8 hours. Moreover, the inference time for the complete network is also much lower, by a factor of almost 1000 (0.296s vs 214s).

3.2 Curve Regression by Bernstein polynomials

There is a high interdependence of the features we selected in Equation (5), since all these features can be expressed in term of ρ_e. Furthermore, it is confirmed by data exploration that ρ_e is the prominent feature for occupancy prediction (and in turn latency prediction), as exemplified in Figure 3.

It is then tempting to try to further simplify our features space and estimate the occupancy from a non-linear transformation of the single feature ρ_e, as:

$$\hat{y} = g(\rho_e) \tag{9}$$

where $\hat{y}$ is the estimate of the occupancy y. The concerns are of course to define simple and efficient functions g, with a low number of parameters, that can model the kind of growth shown in Figure 3, and of course to check that the performance remains interesting.

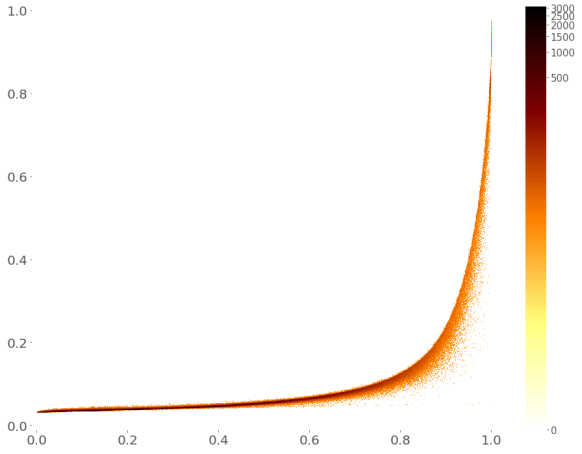


Figure 3: Data of ITU Challenge 2021 [12], ρ_e vs queue occupancy. Color-scale is an indicator of points cloud density.

The estimator g is defined as a linear combination of simple functions f_n:

$$\hat{y} = g(\rho_e) = \sum_n \theta_n \cdot f_n(\rho_e) \tag{10}$$

which is also a linear model in terms of function $f_n(\rho_e)$.

Several solutions were considered in [6] to define or choose the functions f_n. Since we know that the Bernstein polynomials form a basis in the set of polynomial in the interval $[0;1]$; and that the approximation of any continuous function on $[0;1[$ by a Bernstein polynomial converges uniformly, we were led to these polynomials:

$$f_n^K(x) = \binom{K}{n} x^n (1-x)^{K-n} \tag{11}$$

where K is maximum order of polynomials.

As mentioned, (10) can be rewritten as the linear model

$$\hat{y} = g(\rho_e) = \sum_n \theta_n \cdot f_n(\rho_e) = \boldsymbol{\theta}^T \mathbf{u} \tag{12}$$

with $\boldsymbol{\theta}^T = [\theta_0, \ldots, \theta_K]$ and $\mathbf{u}^T = [f_0^K(\rho_e), f_1^K(\rho_e), \ldots, f_K^K(\rho_e)]$. Hence, we have the same form as in (8) for the solution.

In term of performances, we also obtained a minor degradation in MAPE (1.68%) compared to state-of-the-art (1.29%), while improving by several orders the wall training and inference times (2min/3.14s vs 8hrs/214s); though a bit less than the simple linear regression.

4 Adaptive versions

We place ourselves in the context where we have regular snapshots of the state of the network, which allows us to both monitor the quality of predictions, and to track changes in the network. For the n-th series of measurements, let us denote $y(n)$ the measured latency and $\mathbf{u}(n)$ the features. We can also group several snapshots or several links into a vector of latencies $\mathbf{y}(n)$ and matrix $\mathbf{U}(n)$. In the following we will derive equations for this block case, which includes immediately the scalar case.

The minimum mean square error (7) which has the explicit solution (8) can also be solved by a gradient algorithm as

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu \, \nabla J(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}, \tag{13}$$

$$= \boldsymbol{\theta}_k - \mu \left((\mathbf{R}_{uu} + \alpha \mathbf{1}) \, \boldsymbol{\theta}_k - \mathbf{R}_{yu} \right). \tag{14}$$

In (14), we can substitute the true values with estimated ones. In order to introduce adaptivity to context changes in the network, these estimates will preserve the temporal dimension. We thus use either a sliding average

$$\begin{cases} \hat{\mathbf{R}}_{uu}(n) = \sum_{l=0}^{L} \mathbf{U}(n-l)\mathbf{U}(n-l)^T \\ \hat{\mathbf{R}}_{yu}(n) = \sum_{l=0}^{L} \mathbf{U}(n-l)\mathbf{y}(n-l) \end{cases} \tag{15}$$

or an exponential mean

$$\begin{cases} \hat{\mathbf{R}}_{uu}(n) = \sum_{l=0}^{n} \lambda^{l-n} \mathbf{U}(l)\mathbf{U}(l)^T = \lambda \hat{\mathbf{R}}_{uu}(n-1) + \mathbf{U}(n)\mathbf{U}(n)^T \\ \hat{\mathbf{R}}_{yu}(n) = \lambda \hat{\mathbf{R}}_{yu}(n-1) + \mathbf{U}(n)\mathbf{y}(n). \end{cases} \tag{16}$$

where $\lambda \leq 1$ is the forgetting factor.

In the limit case where we take either $L = 0$ or $\lambda = 0$ in the previous formulas, we get the 'instantaneous estimates'

$$\begin{cases} \hat{\mathbf{R}}_{uu}(n) = \mathbf{U}(n)\mathbf{U}(n)^T \\ \hat{\mathbf{R}}_{du}(n) = \mathbf{U}(n)\mathbf{y}(n). \end{cases} \tag{17}$$

we obtain

$$\boldsymbol{\theta}(n+1) = (1 - \mu\alpha)\boldsymbol{\theta}(n) - \mu \mathbf{U}(n) \left(\mathbf{U}(n)^T \boldsymbol{\theta}(n) - \mathbf{y}(n) \right) \tag{18}$$

which reduces to the well known LMS algorithm [14] in the scalar case and no regularization, $\alpha = 0$.

Alternatively, one can try to solve the normal equation (8), using the time dependent estimates as the exponential mean (16). The difficulty with the solution

$$\hat{\boldsymbol{\theta}}(n+1) = \left[\hat{\mathbf{R}}_{uu}(n+1) + \alpha \mathbf{1} \right]^{-1} \hat{\mathbf{R}}_{yu}(n+1) \tag{19}$$

is the inversion, for each n, of the correlation matrix. Let us denote

$$\mathbf{K}(n+1) = \left[\hat{\mathbf{R}}_{uu}(n+1) + \alpha \mathbf{1} \right]^{-1}. \tag{20}$$

Using (16), we have

$$\mathbf{K}(n+1)^{-1} = \lambda \hat{\mathbf{R}}_{uu}(n) + \mathbf{U}(n+1)\mathbf{U}(n+1)^T + \alpha \mathbf{1} \tag{21}$$

$$= \lambda \left(\hat{\mathbf{R}}_{uu}(n) + \alpha \mathbf{1} \right) + \mathbf{U}(n+1)\mathbf{U}(n+1)^T + \alpha(1-\lambda)\mathbf{1} \tag{22}$$

$$= \lambda \mathbf{K}(n)^{-1} + \mathbf{U}(n+1)\mathbf{U}(n+1)^T + \alpha(1-\lambda)\mathbf{1} \tag{23}$$

and

$$\mathbf{K}(n+1) = \left[\left(\lambda \mathbf{K}(n)^{-1} + \mathbf{U}(n+1)\mathbf{U}(n+1)^T\right) + \alpha(1-\lambda)\mathbf{1}\right]^{-1} \tag{24}$$

$$= \left[\mathbf{Q}(n+1) + \delta\mathbf{1}\right]^{-1} \tag{25}$$

with

$$\mathbf{Q}(n+1) = \left(\lambda \mathbf{K}(n)^{-1} + \mathbf{U}(n+1)\mathbf{U}(n+1)^T\right) \tag{26}$$

and $\delta = \alpha(1-\lambda)$ The matrix inversion lemma enables to reduce the inversion of $\mathbf{Q}(n+1)$ to

$$\begin{aligned}\mathbf{Q}(n+1)^{-1} =& \frac{1}{\lambda}\mathbf{K}(n) - \frac{1}{\lambda^2}\mathbf{K}(n)\mathbf{U}(n+1)\times \\ & \left(\mathbf{1} + \frac{1}{\lambda}\mathbf{U}(n+1)^T\mathbf{K}(n)\mathbf{U}(n+1)\right)^{-1} \mathbf{U}(n+1)^T\mathbf{K}(n),\end{aligned} \tag{27}$$

which simplifies to

$$\mathbf{Q}(n+1)^{-1} = \frac{1}{\lambda}\mathbf{K}(n) - \frac{1}{\lambda^2}\frac{\mathbf{K}(n)\mathbf{u}(n+1)\mathbf{u}(n+1)^T\mathbf{K}(n)}{1 + \frac{1}{\lambda}\mathbf{u}(k+1)^T\mathbf{K}(n)\mathbf{u}(k+1)}, \tag{28}$$

for scalar observations. Now, we can use the Taylor expansion to get

$$\mathbf{K}(n+1) = \left[\mathbf{Q}(n+1) + \delta\mathbf{1}\right]^{-1} = \mathbf{Q}(n+1)^{-1} - \delta\mathbf{Q}(n+1)^{-2} + \delta^2\mathbf{Q}(n+1)^{-3} + \dots \tag{29}$$

This gives us a way to compute recursively the inverse of the regularized estimate of the correlation matrix by combining (27) and (29) into

$$\mathbf{K}(n+1) \approx \mathbf{Q}(n+1)^{-1} - \delta\mathbf{Q}(n+1)^{-2} \tag{30}$$

which, by (27), does not require the inversion of $K(n)$.

In both cases, we have the updating formula

$$\boldsymbol{\theta}(n+1) = \boldsymbol{\theta}(n) + \mathbf{K}(n+1)\mathbf{U}(n+1)[\mathbf{y}(n+1) - \boldsymbol{\theta}(n)^T\mathbf{U}(n+1)]. \tag{31}$$

5 Experiments and results

5.1 Dataset

We generate a dataset thanks to a public challenge data generator [12]. This data generator is based on the well-known OMNET++ discrete event network simulator[13]. The published simulator is available as a docker image. However, due to the rules of the 2022 edition of the challenge, it is not possible to generate large topologies, i.e. no more than 10 nodes. Since our models are link-based, the use of small topologies does not seem to be a problem. The simulator is parameterized by a traffic matrix and a topological graph that are easy to generate thanks to the provided API.

Our generated dataset, used is this paper, is the result of 11900 simulations of the same topology graph of 10 nodes and 30 links, subject to 100 different traffic matrices. In order to get complex results of simulations but at low cost, we made the choice to model a network with small queue buffers (8000 bits) and possibly subject of high traffic intensities (maximum traffic rate set to 4000 bits/s for each flow). Then for each traffic matrices, we alter the capacity of the network according to a sigmoid, in order to model a network subject to jamming, with 2 stationary states. The proposed jamming may cause a decrease in the capacity of the network links by up to a factor of 5, as depicted on Figure 4a. For simplification purposes, we have considered that each link of the network has the same capacity. This result in a U-shaped distribution of our link data samples according to the link capacity as shown in Figure 4b.

5.2 Results

From the generated data, we validate our approach along several axes.

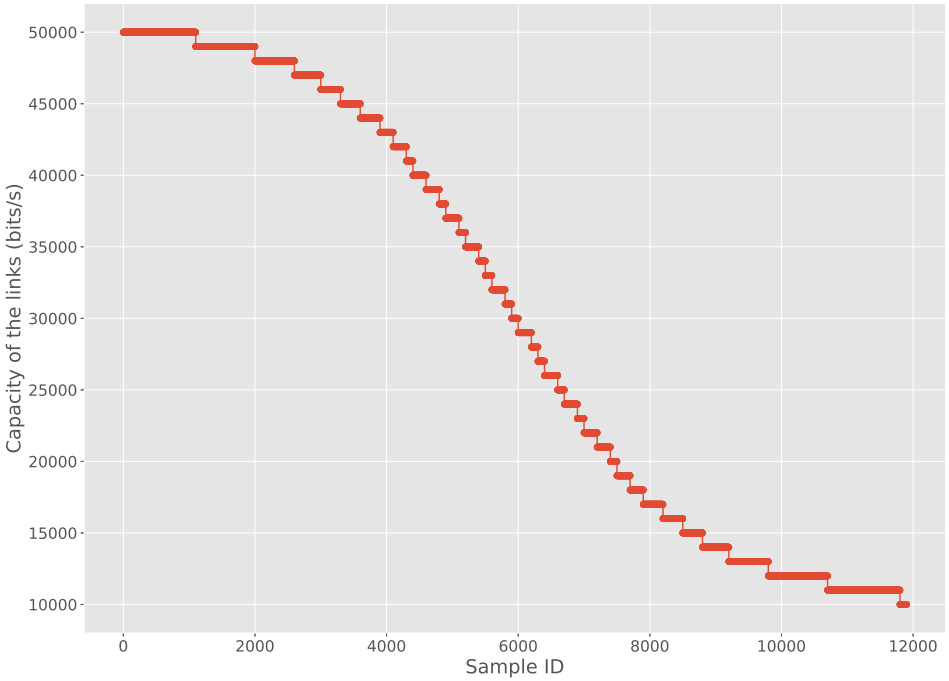
5.2.1 Global performances

First, we establish the benchmark performances based on the global methods presented in the paper [6] and Sections 3.1 and 3.2.

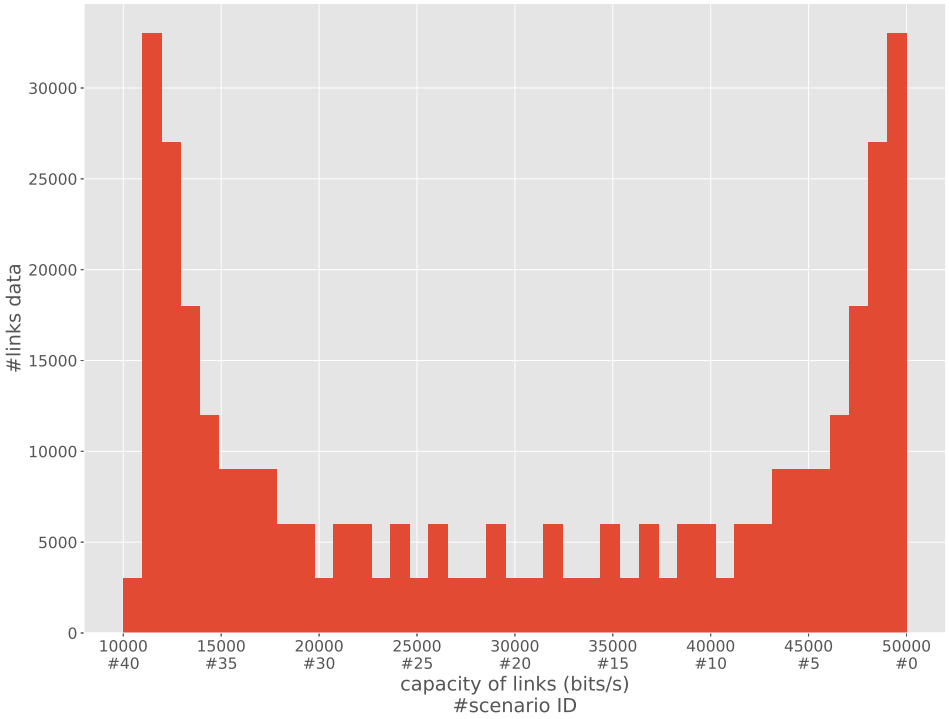
For the linear regression method described in Section 3.1, we obtain an MSE of 5.86e-4 and a MAPE of 9.58% for the queue occupancy prediction and an MSE of 1.10e-3 and a MAPE of 10.26%. for the end-to-end latency prediction.

Concerning the curve regression using Bernstein polynomials (of degree 8) described in in Section 3.2, we obtain an MSE of 4.52e-4 and a MAPE of 8.72% for the queue occupancy prediction and an MSE of 9.35e-4 and a MAPE of 9.95% for the end-to-end latency prediction.

Note that these benchmark performances are below the performances obtained in [6], but the dataset we use here is probably more severe since using ground truth value for occupancy results in an MSE 6.03e-4 of a MAPE 9.34% for the flow delay prediction. That is indeed very close of the obtained results.



(a) Capacity alteration to model jamming with a decrease of the capacity up to a factor of 5.



(b) Link capacity distribution of the generated dataset.

Figure 4: Overview of the generated dataset

5.2.2 Behavior of iterative algorithms

In a second step, we verify that the algorithms presented in section 4 converge and allow us to recover these performances. With a forgetting factor of 1 (use of all data with the same weight) and a block size of 10, we observe, for example in Figure 5, that the model coefficients converge towards a stable value, and that MAPE recovers the value obtained with the global method using all data. The convergence is obtained in less than 10,000 operations. It is thus possible to replace the global method, which is already low-cost, with an approach where the calculations are carried out recursively.

5.2.3 Adaptivity

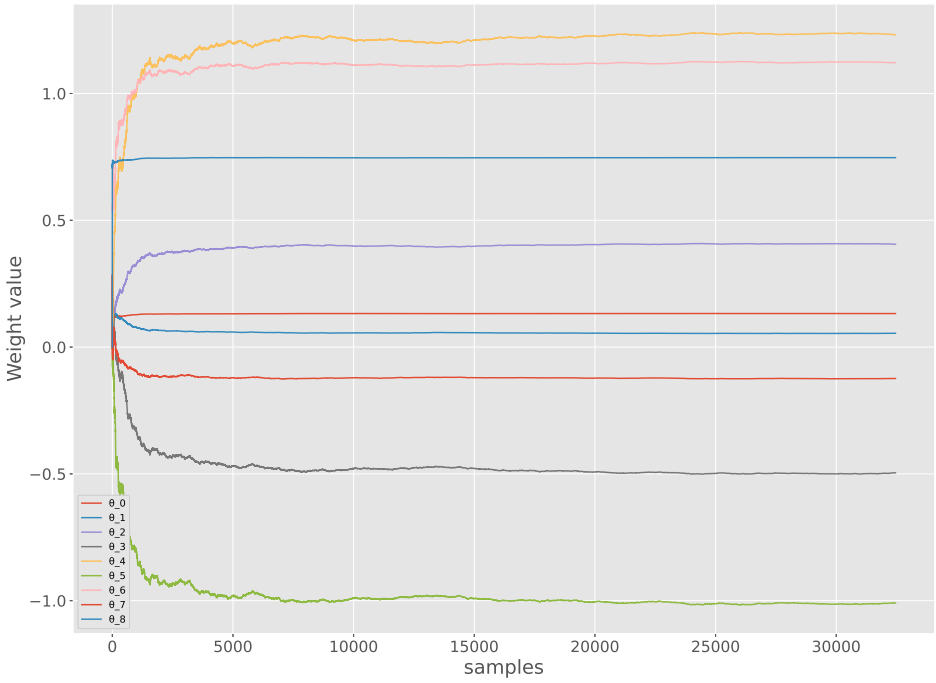
In a third step, we compare the algorithms to the case of network modifications We consider an abrupt change in the network capacity, which could correspond to a jamming scenario. We then examine how the two adaptive algorithms presented (linear regression with judiciously chosen features; and Bernstein polynomial model) can detect and adapt to these modifications. In this context, we examine the role of the forgetting factor and the regularization parameter. Figure 6 and Figure 7 present the results for the case of a capacity change. We observe that (i) the square of the residual error, smoothed over 100 points, is a remarkable indicator of a change in the network; and (ii) that the model coefficients readjust over the iterations after this change.

5.2.4 Discussion

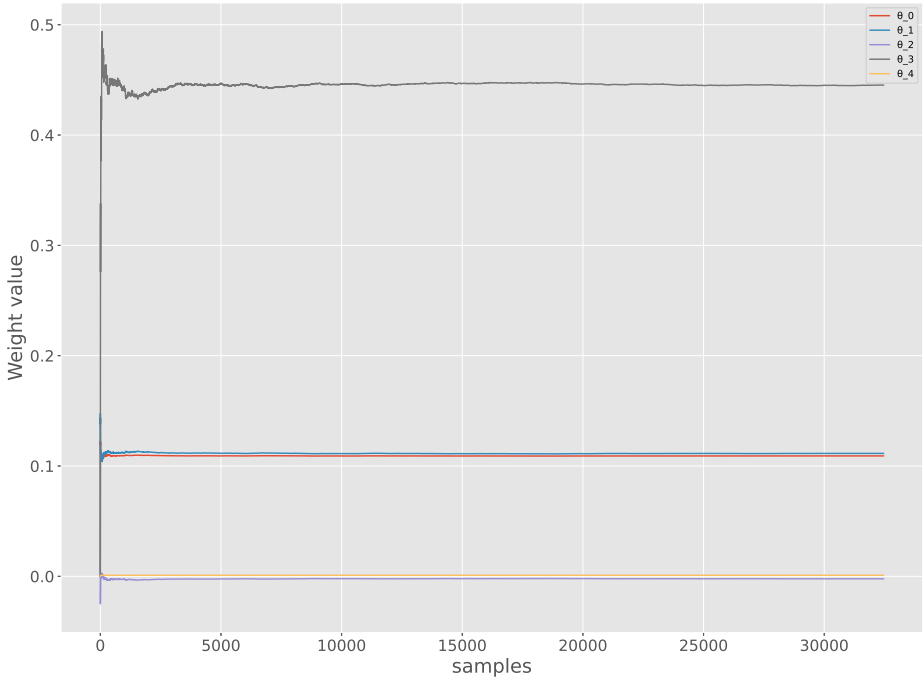
These experiments show the effectiveness and relevance of our iterative and adaptive versions of end-to-end latency estimation procedures. The iterative versions have the same performance as their global counterparts; an even lower cost since they can be implemented iteratively as the data is received or made available. The convergence time for the model coefficients is a few thousand samples while the global model used around 350,000 samples for training, while the GNN models require several million samples. Moreover, we observe that the residual error converges very quickly, in some tens of samples, which means that although the convergence of the models' coefficients does not seem to be complete, they are equivalent from the point of view of performance for occupancy prediction. From an operational point of view, the model can be refreshed regularly, and the predicted KPIs between these updates can be used for intelligent routing. As we have observed, residual error monitoring is an excellent indicator of changes in the network state.

6 Conclusion

In this paper, we considered the problem of designing efficient and low-cost algorithms for KPI prediction that are locally implementable and adaptive to network changes. Based on a previous work, we have argued and developed adaptive solutions, introducing in addition a regularization term in order to stabilize the results. We used a public domain simulator to simulate networks and generate relevant data. The experiments demonstrate the effectiveness and relevance of these algorithms. Thus, we now have low-complexity models that can be implemented iteratively at the level of local links. We have the possibility to predict the occupancy of the different links, and the end-to-end latencies (the models predict the occupancy of the queues, then compute analytically the delay for each link and finally aggregate along the path). Moreover, the adaptability of the solution allows to follow changes in the network state, always at a minimal cost, by re-adapting from the current solution and new data. The continuation of the work will focus on the choice criteria of the forgetting factor, on the impact of the regularization factor, in order to find automatic selection methods. Of course, the approaches considered here will have to be considered and adapted for other types of KPI, such as error rate or jitter.

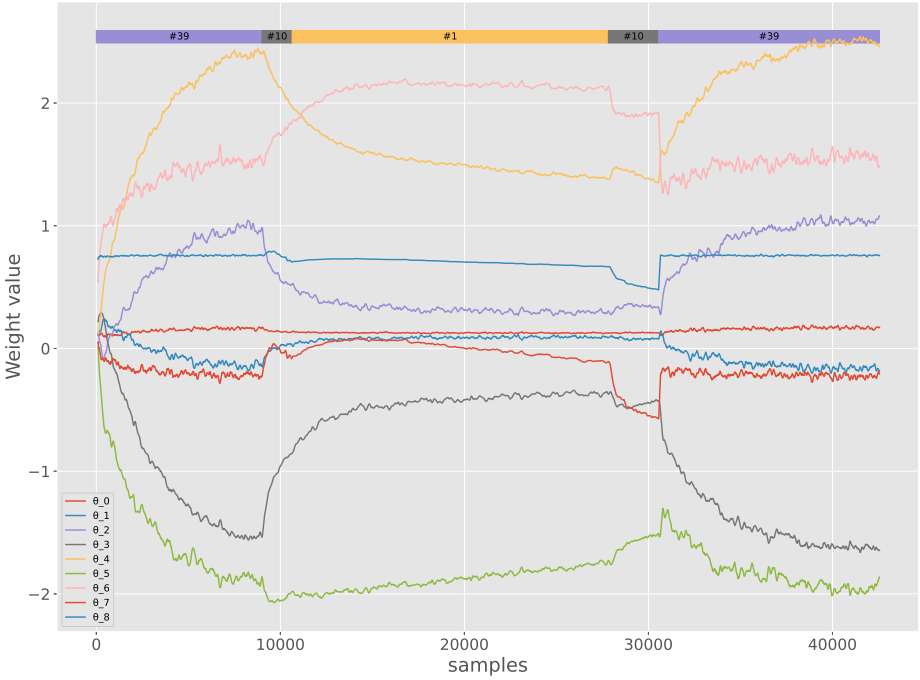


(a) Iterative curve-fitting based on Bernstein polynomials of degree 8.

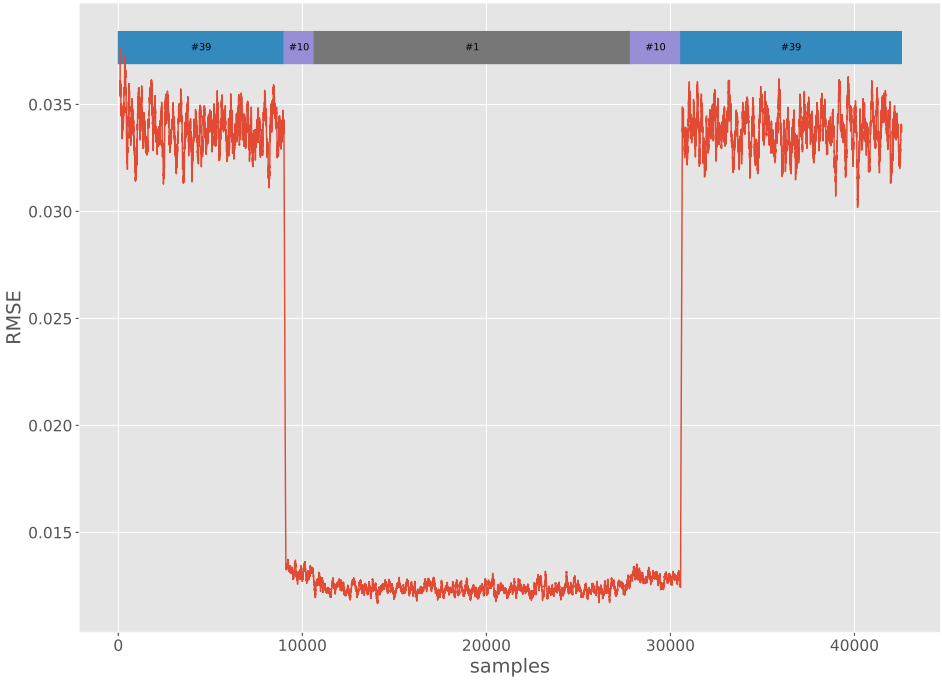


(b) Iterative Linear Regression

Figure 5: Evolution of weights for our iterative methods without forgetting (non-adaptive) while fitting the whole dataset.

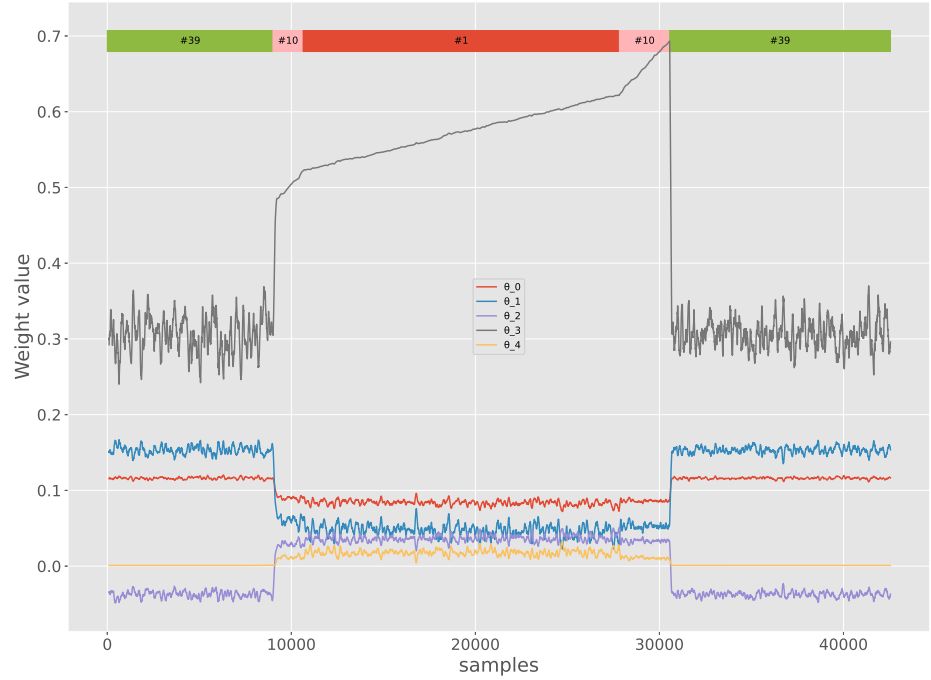


(a) Evolution of weights along the scenario.

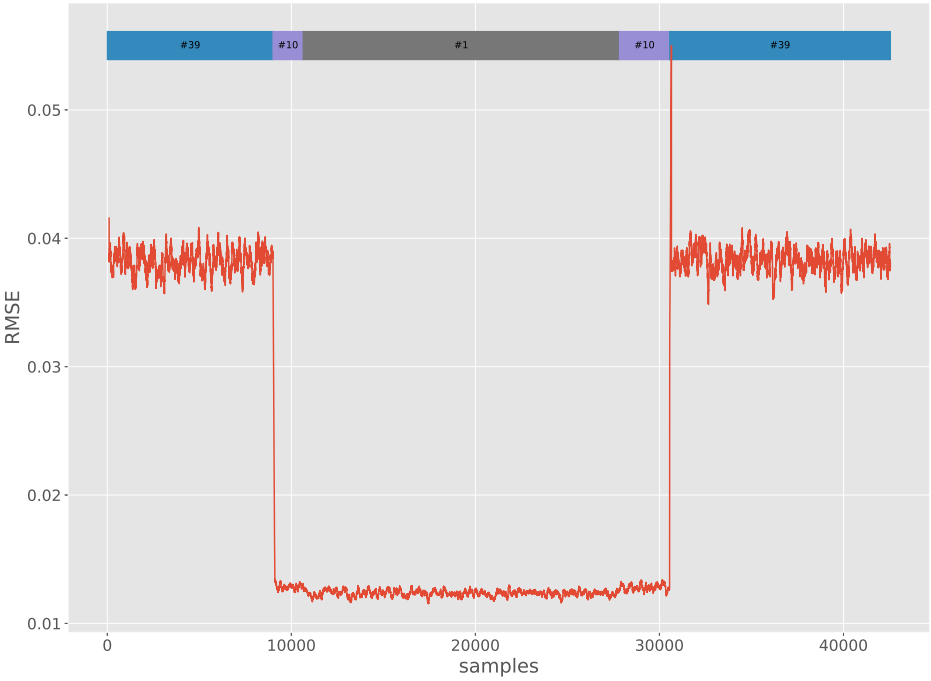


(b) Evolution of the RMSE along the scenario.

Figure 6: Evolution of weights and $\sqrt{\text{MSE}}$ (RMSE) for our adaptive approach of Bernstein polynomial curve regression of degree 8,$\lambda = 0.9$, $\alpha = 0.08$. Scenario describes a nominal period between 2 periods of jamming. #39 corresponds to a link capacity of 11 Kbits/s, #10 40 Kbits/s and #1 49 Kbts/s. Figure is smoothed over 100 points.



(a) Evolution of weights along the scenario.



(b) Evolution of the RMSE along the scenario.

Figure 7: Evolution of weights and $\sqrt{\text{MSE}}$ (RMSE) for our adaptive approach of Linear Regression,$\lambda = 0.9$, $\alpha = 0.08$. Scenario describes a nominal period between 2 periods of jamming. #39 corresponds to a link capacity of 11 Kbits/s, #10 40 Kbits/s and #1 49 Kbts/s. Figure is smoothed over 100 points.

References

[1] Amin, R., Reisslein, M., Shah, N.: Hybrid SDN networks: A survey of existing approaches. IEEE Communications Surveys & Tutorials **20**(4), 3259–3306 (2018)

[2] de Aquino Afonso, B.K.: GNNet challenge 2021 report (1st place). `https://github.com/ITU-AI-ML-in-5G-Challenge/ITU-ML5G-PS-001-PARANA` (2021)

[3] Barcelona Neural Networking Center: The graph neural networking challenge 2020. `https://bnn.upc.edu/challenge/gnnet2020`

[4] Chua, F.C., Ward, J., Zhang, Y., Sharma, P., Huberman, B.A.: Stringer: Balancing latency and resource usage in service function chain provisioning. IEEE Internet Computing **20**(6), 22–31 (2016)

[5] Jahromi, H.Z., Hines, A., Delanev, D.T.: Towards application-aware networking: Ml-based end-to-end application KPI/QoE metrics characterization in SDN. In: 2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN). pp. 126–131. IEEE (2018)

[6] Larrenie, P., Bercher, J.F., Lahsen-Cherif, I., Venard, O.: Low complexity approaches for end-to-end latency prediction. In: Proceedings of the 13th IEEE International Conference On Computing, Communication and Networking Technologies. IEEE (2022)

[7] Pasca, S.T.V., Kodali, S.S.P., Kataoka, K.: AMPS: Application aware multipath flow routing using machine learning in SDN. In: 2017 Twenty-third National Conference on Communications (NCC). pp. 1–6. IEEE (2017)

[8] Poularakis, K., Iosifidis, G., Tassiulas, L.: SDN-enabled tactical ad hoc networks: Extending programmable control to the edge. IEEE Communications Magazine **56**(7), 132–138 (2018)

[9] Poularakis, K., Qin, Q., Nahum, E.M., Rio, M., Tassiulas, L.: Flexible SDN control in tactical ad hoc networks. Ad Hoc Networks **85**, 71–80 (2019)

[10] Rusek, K., Suárez-Varela, J., Mestres, A., Barlet-Ros, P., Cabellos-Aparicio, A.: Unveiling the potential of graph neural networks for network modeling and optimization in SDN. In: Proceedings of the 2019 ACM Symposium on SDN Research. pp. 140–151 (2019)

[11] Singh, S., Jha, R.K.: A survey on Software Defined Networking: Architecture for next generation network. Journal of Network and Systems Management **25**(2), 321–374 (2017)

[12] Suárez-Varela, J., et al.: The graph neural networking challenge: a worldwide competition for education in AI/ML for networks. ACM SIGCOMM Computer Communication Review **51**(3), 9–16 (2021)

[13] Varga, A., Hornig, R.: An overview of the omnet++ simulation environment. In: 1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems (2010)

[14] Widrow, B., Stearns, S.: Adaptive Signal Processing. Edited by Alan V. Oppenheim, Prentice-Hall (1985)