QT-ROUTENET: IMPROVED GNN GENERALIZATION TO LARGER 5G NETWORKS BY FINE-TUNING PREDICTIONS FROM QUEUEING THEORY

Bruno Klaus de Aquino Afonso[1] and Lilian Berton[1]
[1]Federal University of São Paulo (ICT-UNIFESP), Brazil

NOTE: Corresponding author: Bruno Klaus de Aquino Afonso, bruno.klaus@unifesp.br

Abstract – *In order to promote the use of machine learning in 5G, the International Telecommunication Union (ITU) proposed in 2021 the second edition of the ITU AI/ML in 5G challenge, with over 1600 participants from 82 countries. This work details the second place solution overall, which is also the winning solution of the Graph Neural Networking Challenge 2021. We tackle the problem of generalization when applying a model to a 5G network that may have longer paths and larger link capacities than the ones observed in training. To achieve this, we propose to first extract robust features related to Queueing Theory (QT), and then fine-tune the analytical baseline prediction using a modification of the Routenet Graph Neural Network (GNN) model. The proposed solution generalizes much better than simply using Routenet, and manages to reduce the analytical baseline's 10.42 mean absolute percent error to 1.45 (1.27 with an ensemble). This suggests that making small changes to an approximate model that is known to be robust can be an effective way to improve accuracy without compromising generalization.*

Keywords – 5G networks, fine-tuning, graph neural network, ITU challenge, queueing theory

1. INTRODUCTION

During the year of 2021, the International Telecommunication Union (ITU) once again brought to the forefront the use of machine learning as a means to maximize the efficiency of 5G. The second edition of the *"ITU AI/ML in 5G challenge"* introduced a diverse set of challenges related to the development and training of machine learning models to solve particular problems within the realm of 5G networks. Over 1600 competitors from 82 countries were asked to solve problems that were put forth by hosts from different regions [1]. This work details the first place solution of the challenge proposed by the Barcelona Neural Networking Center, named *Graph Neural Networking Challenge 2021 - Creating a Scalable Digital Network Twin*, a.k.a. **GNNet Challenge 2021**. This solution would later on compete against winning solutions from other regional hosts in the Grand Challenge Finale, ending up with second place overall.

Much like in the previous year, the GNNet challenge was centered around creating a predictive model for 5G networks: given a topology and routing configuration, one must predict the per-path-delay. In addition, the GNNet Challenge 2021 had a specific goal in mind: to address the current limitations of Graph Neural Network (GNN) architectures, whose generalization suffers greatly when predicting on larger graphs. This was verified by the organizers to be the case for *RouteNet* [2], a message-passing GNN model that influenced most solutions from the 2020 edition of the challenge [3]. When creating a dataset, it is usually not feasible to gather data from a currently deployed network, as that would require us to explore edge cases that directly lead to service disruption, such as link failures. The alternative is to generate everything from a small network testbed created in the vendor's lab. The distribution of the graphs observed in validation/test set are therefore different from the ones observed in training.

Table 1 – Types of approaches

	Fast Enough?	Top tier results on small graph?	Generalizes to larger graphs?
Analytical	✓	✗	✓
Packet simulators	✗	✓	✓
RouteNet	✓	✓	✗
Proposed solution	✓	✓	✓

To understand the solution detailed in this report, it is helpful to look at previous approaches (Table 1). They are divided into 3 categories: analytical approximations, packet simulators and RouteNet, a model based on message-passing GNNs. Packet simulators were not allowed in the competition in principle due to excessive running times; analytical approaches generalize well and run fast, but they do not offer competitive performance; RouteNet is still fast and more accurate than analytical approaches, but fails to generalize to larger graphs. For our proposed solution, we **extract invariant features from the analytical approach**, and feed them to a GNN. This way, we can **maintain generalization while outperforming the purely analytical** approach.

2. RELATED WORK

The problem of predicting traffic in networks has been long studied within Queueing Theory (QT) [4], a branch of mathematics that deals with the analysis of waiting lines. In a queueing system, customers randomly arrive at a cer-

©International Telecommunication Union, 2022
Some rights reserved. This work is available under the CC BY-NC-ND 3.0 IGO license: https://creativecommons.org/licenses/by-nc-nd/3.0/igo/.
More information regarding the license and suggested citation, additional permissions and disclaimers is available at:
https://www.itu.int/en/journal/j-fet/Pages/default.aspx

tain place to receive a service, and then leave upon its completion. By modeling the arrival process and service of customers with probability distributions, we can use QT to estimate Key Performance Indicators (KPIs) such as delay and jitter.

Within the context of 5G networks, we are interested in modeling the arrival and service of network packets. We must model each link of the network as a separate queue. The simplest case is the M/M/1 queue, where the arrival process is Poisson, the service process is exponential, and there is one server. More intricate models include the M/M/1/B queue, which has a buffer that can hold up to B items.

Traffic flow will be heavily dependent on the routing algorithm and network topology. Given a model of each link as a queue, one can derive a system of equations related to traffic balance on the network. By solving those equations, one arrives at the analytical solution for the relevant performance metrics. This provides us a way to analyze these systems with a solid theoretical foundation. However, analytic models used to predict KPIs in large-scale networks often make unrealistic assumptions about the network, and as a result are not accurate enough.

If the time required to compute the KPI estimate is not a concern, it is interesting to consider packet simulators such as OMNeT++[5]. A model in OMNeT++ consists of nested modules that communicate by message passing. The topology of the model is specified by a topology description language. In this setting, analytical intractability is not a concern, and one can get more accurate results by simulating individual packets. However, this comes at a high cost when you consider the running time. According to the organizers of the GNNet Challenge 2021, packet simulators were used to help create the competition's dataset. Combined with their excessive running times, it is no surprise that they were prohibited.

Machine learning is a powerful tool that can help us achieve better results than we would get through analytical methods alone. By using deep neural networks, we can learn the intricacies of real-world networks by leveraging huge amounts of data. As the input of our model is a network, the problem is very suitable to graph neural networks [6].

Routenet [2] is the machine learning approach most influential to our work. It is a GNN-based model that uses update functions to maintain and update representations during message-passing iterations between links and paths. In addition to working with a natural network representation, Routenet is able to relate topology, routing, and input traffic in order to accurately estimate KPIs. Routenet outperforms the analytical baseline even when the latter was particularly suited to the dataset. In addition, it can handle different network topologies than the ones observed in training.

After the initial promising results, more experimentation was conducted in [7] to evaluate the generalization capabilities of Routenet. The authors found that, when the evaluation data is drastically different from the training data, Routenet's predictions get significantly worse. This includes, but is not limited to, larger link capacities and longer paths.

During the Graph Neural Networking Challenge 2021, many approaches were devised to solve this generalization problem. At the time of writing, one available example is the data augmentation of [8], where link capacities are defined as a product of a virtual reference link capacity and a scaling factor.

3. FRAMEWORK

Our model was built from scratch in the Python language using Pytorch 1.8.1 [9] and Pytorch Geometric 1.7.0 [10]. The code requires a GPU; we used an RTX 3080 with 10GB VRAM. Moreover, 16GB of RAM is enough to not run out of memory. Our code was divided into 3 different Jupyter notebooks: one for creating the dataset, two for **2 similar models** whose **average** constituted the final prediction used for this challenge. The source code and frozen model weights are available on Github [1].

3.1 Input format

Table 2 – Converted dataset information

	Samples	Network size
Training	120000	25-50 nodes
Validation	3120	51-300 nodes
Test	1560	51-300 nodes

Because we are interested in generalization to larger networks, the samples in the validation dataset are considerably larger. As shown in Table 2, the networks seen in training have at most 50 nodes, whereas those in the validation set may have up to 300.

The validation set is divided into 3 subsets. All subsets were provided to participants by the competition organizers, and each one captures a type of network that differs from the training set. In Subset 1, longer paths are artificially generated, and only those paths transmit traffic. Subset 2 uses variants of a shortest path routing policy, with all source-destination paths producing traffic. To make up for the increased traffic, the routing includes links with larger capacity than the ones encountered in the training data. Lastly, Subset 3 can be considered a combination of the previous two: it uses routing schemes with larger paths, and also larger link capacities. The test set is assumed to have the same distribution as the validation set.

[1] https://github.com/ITU-AI-ML-in-5G-Challenge/ITU-ML5G-PS-001-PARANA

In order to improve speed when training our model, we modify the script provided to challenge participants[2]. Our script allows one to run multiple processes in parallel, to speed up the creation of this "converted" dataset. Due to the large amount of files, we recommend an SSD with at least 60GB of memory available. Each flow between two nodes is considered as a separate entity, with the following attributes:

1. `p_AvgPktsLambda`: Average number of packets of average size generated per time unit.
2. `p_EqLambda` Average bit rate per time unit.
3. `p_AvgBw`: Average bandwidth between nodes (bits/time unit).
4. `p_PktsGen`: Packets generated between nodes (packets/time unit).
5. `p_TotalPktsGen`: Total number of packets generated during the simulation.

We zero out `p_TotalPktsGen`, as we did not find the inclusion of simulation time helpful. In order to subject the network to **more varied input values**, we divide link capacity `l_LinkCapacity` (the sole link attribute) and all path attributes by `p_AvgPktsLambda` before normalization.

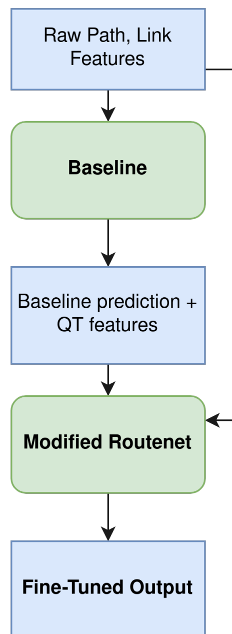


Fig. 1 – Basic steps taken in our model. Instead of directly using the original "raw" path and link features, we feed them to the QT baseline and extract higher-level features, including a reliable approximate prediction. A modified Routenet fine-tunes this prediction to improve the QT baseline while still maintaining generalization.

[2] `https://github.com/BNN-UPC/GNNetworkingChallenge/tree/2021_Routenet_TF`

3.2 QT-Routenet: Model overview

The idea behind the proposed QT-Routenet model is very simple in principle, combining Routenet and smart feature extraction. In Fig. 1, we can see that the original features are fed to the baseline, resulting in higher-level QT features, e.g. the baseline prediction. The higher-level features (and, optionally, the original features) are then fed to a modified Routenet to obtain fine-tuned output.

By experimentation, it was verified by the organizers that simply feeding the raw path and link features to the existing Routenet architecture leads to over 300% **Mean Absolute Percent Error (MAPE)** in the competition dataset [11]. This is likely because the distribution of those features in the validation and test sets is much different than the one observed in training. On the other hand, the baseline prediction remains reasonably consistent between the training set and all validation sets. This motivated us to tackle the problem at the input level: using the baseline prediction (and other QT features), we wanted to stay as conservative and close to the baseline as possible, but still use a graph neural network to obtain improved results.

One could potentially describe our approach as using queueing theory to assign an initial (imperfect) label, and then learning to smooth it adequately over the network. However, we should always refer to the baseline prediction as a feature, not a label. The reason is twofold: first, the actual label in this problem is the path delay obtained in the simulation conducted by the organizers; secondly, the prediction will be available for all future data, which avoids some overfitting issues when using known labels to optimize graph models with gradient descent [12].

3.3 Defining the heterogeneous graph

Our GNN model works with a matrix $\boldsymbol{X}$, which is the concatenation of all attributes. The number of rows is equal to the sum of the number of paths, links and nodes. These columns all remain fixed, so we also need to add "hidden state" columns to each of these entities. These columns are used to perform message-passing, taking in information about other "hidden state" columns and also fixed columns. We denote by $\boldsymbol{X}_P, \boldsymbol{X}_L, \boldsymbol{X}_N$ the fixed columns of paths, links, and nodes. The hidden state columns are zero-initialized and denoted by $\boldsymbol{X}_{Ph}, \boldsymbol{X}_{Lh}, \boldsymbol{X}_{Nh}$. We intended to put some provided global attributes into $\boldsymbol{X}_N$ but eventually opted for setting them to zero. We standardize all features before feeding them to the model.

Next, we must go over network topology. We denote the topology by $\boldsymbol{E}$. The conditions for the existence of edges are:

- $\boldsymbol{E}_{PL}$: Whenever a link is part of a path
- $\boldsymbol{E}_{PN}$: Whenever a node is part of a path
- $\boldsymbol{E}_{LN}$: Whenever a node is part of a link

© International Telecommunication Union, 2022

In practice, edges are directed. We may use the terminology $\boldsymbol{E}_{PL}$ to indicate that the direction is *path-to-link*, whereas $\boldsymbol{E}_{LP}$ is *link-to-path*. In addition, our code contains a special function, `SeparateEdgeTimeSteps`, which is able to output a list separating $\boldsymbol{E}_{LP}$. The k-th element of this list has all of the $\boldsymbol{E}_{LP}$ edges satisfying a condition: that the link is the k-th one found while traversing the path. This separation allows us to preserve order information and use recurrent layers.

3.4 Message passing model

There are two message passing models, which are listed as Algorithm 1 and Algorithm 2 **(see Appendix)**. The main difference is that the first model includes nodes in the message mechanism, which are ignored by the second model. For both models, we feed the initial input matrix $\boldsymbol{X}$ to a **Multilayer Perceptron (MLP)**. Then, we perform a number of message-passing iterations using convolutions. First the path entities receive messages, then nodes, and lastly links.

Messages are exchanged from links to paths using a single **Chebyshev Graph Convolutional Gated Recurrent Unit Cell** [13] layer imported from Pytorch Geometric Temporal [14]. All other convolutions were set to be **Graph Attention (GAT)** [15] layers, and different GAT layers are used for each iteration. We set the first few hidden columns to be equal to the baseline features, so that this information is preserved similarly to the other fixed features. After the message-passing rounds, we feed $\boldsymbol{X}_L$ and $\boldsymbol{X}_{Lh}$ to another MLP to obtain the prediction for *average queue utilization*. Finally, the average path delay is obtained using the formula

$$\texttt{pathDelay} \approx \sum_{i=0}^{\texttt{n_links}} \texttt{delayLink}(i) \tag{1}$$

The delay on each link includes both the time waiting in the queue, as well as the time actually passing through the link. The former is given as

$$\texttt{queue_delay}_i = \frac{\texttt{avg_utilization}_i \times \texttt{queue_size}_i}{\texttt{link_capacity}_i} \tag{2}$$

whereas the latter can be approximated as:

$$\texttt{transmission_delay}_i = \frac{\texttt{mean_packet_size}}{\texttt{link_capacity}_i} \tag{3}$$

From there, we calculate the average utilization of the link:

$$\texttt{avg_utilization}_i = \sum_{j=0}^{b_i} j(\pi_0 \rho_i^j) \tag{4}$$

3.5 Extracting queueing theory features

Perhaps the most important aspect of our model is its use of an analytical baseline that serves as a feature extraction step. This algorithm (Algorithm 3) is based on *Queueing Theory* (QT), and iteratively calculates the traffic on links and blocking probabilities. After these iterations, we calculate the traffic intensity ρ, probability of being in state zero π_0, and predicted average occupancy $\boldsymbol{L}$. The first two are used as features only in the second model. In addition, we also extract the baseline's per-path-delay prediction, using Equation *(1)*.

Our work closely follows the M/M/1/B model used in [16]. For convenience, we list the same formulas used to model the network. Let $\lambda_{k,i}$ be the amount of traffic from some path p_k passing through some link l_i. Each path is a sequence of links, and so we can use the notation $\lambda_{k,k(j)}$ to indicate the traffic of path p_k going through the j-th link encountered while traversing it. The equations governing the network are:

$$\lambda_{k,i} = 0, \qquad \text{if } l_i \notin p_k \tag{5}$$

$$\lambda_{k,k(1)} = A_k \tag{6}$$

$$\lambda_{k,k(j)} = A_k \prod_{i=1}^{j-1} (1 - \texttt{Pb}_{k(i)}) \qquad \text{if } j > 1 \tag{7}$$

$$\texttt{Pb}_i = \frac{(1-\rho_i)\rho_i^{b_i}}{1-\rho_i^{b_i+1}} \tag{8}$$

$$\rho_i = \frac{\sum_k \lambda_{k,i}}{c_i} \tag{9}$$

where: A_k is the demand on the path p_k; b_i is the buffer size on link l_i; $\texttt{Pb}_i$ is the blocking probability on link l_i; ρ_i is the utilization of the link, i.e. the ratio between the total traffic on link l_i and its capacity c_i. The dataset of this competition provides us with b_i, c_i, and A_k. Specifically, we have $\forall i : b_i = B = 32$, i.e. all queues can hold up to 32 packets. On the other hand, we must use a fixed point algorithm to iteratively update our estimates of the traffic $\lambda_{k,i}$ and blocking probabilities $\texttt{PB}_i$.

Once all the previous QT quantities have been estimated, we can compute the probability that there isn't a packet in the link's queue:

$$(\pi_0)_i = \frac{(1-\rho_i)}{1-\rho_i^{b_i+1}} \tag{10}$$

From there, we calculate the average utilization of the link:

$$\texttt{avg_utilization}_i = \frac{1}{B} \sum_{j=0}^{B} j(\pi_0 \rho_i^j) \tag{11}$$

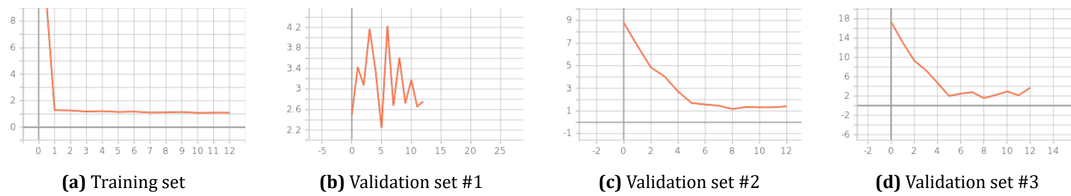
where B is the maximum number of packets per queue. Next, the mean packet size is obtained as

$$\frac{\texttt{queue_size}}{B} \tag{12}$$

where `queue_size` is given as 32000 for this dataset. This means that the total delay (waiting and passing through the link) is:

$$\frac{(x + \sum_{j=0}^{B} j(\pi_0 \rho_i^j))}{B} \times \frac{\texttt{queue_size}}{c_i} \tag{13}$$

© International Telecommunication Union, 2022



(a) Training set **(b)** Validation set #1 **(c)** Validation set #2 **(d)** Validation set #3

Fig. 2 – Model 2 tensorboard run for the training set and validation subsets. The horizontal axis represents the number of epochs, with the vertical axis corresponding to the mean absolute percent error averaged on that epoch. The submission used for the final prediction was the one for epoch #8. Note that the validation data is the same for each step, while a random 10% of the training set is evaluated during an epoch.

with $x = 1$. After some experimentation, we found that substituting $x = \pi_0$ gave slightly better results for this dataset.

3.6 Hyper-parameters

The other significant difference between the two models lies in the model size. When developing the second model, we opted to scale down as much as possible. This can be observed by looking at the size (i.e. number of columns) of $\boldsymbol{X}_{Lh}, \boldsymbol{X}_{Ph}, \boldsymbol{X}_{Nh}$, as well as the hidden layer size for the second multilayer perceptron. The entities that represented individual nodes were entirely discarded, justified partly due to the absence of meaningful attributes for nodes.

Table 3 – Differences between the two models used. Model 2 uses less hidden input columns, and opts for a simple Linear layer instead of an MLP before message passing.

Model	# of hidden input columns	MLP_1	MLP_2
1	$\boldsymbol{X}_{Ph}$:64 $\boldsymbol{X}_{Lh}$:64 $\boldsymbol{X}_{Nh}$:64	Linear(128) LeakyRELU() Linear(3×64) LeakyRELU()	Linear(512) LeakyRELU() Linear(512) LeakyRELU() Linear(1)
2	$\boldsymbol{X}_{Ph}$:8 $\boldsymbol{X}_{Lh}$:8	Linear(2×8)	Linear(128) LeakyRELU() Linear(32) LeakyRELU() Linear(1)

The number of message passing rounds for both models is three. Model 1 uses five baseline iterations, whilst Model 2 reduces that to three iterations. Both models perform message passing iterations for three rounds.

Lastly, we set the initial guess of blocking probabilities to 0.3 when training our models, even though it is most common to set it to zero. This did not seem to affect the convergence of the method.

4. TRAINING AND RESULTS

We use the Adam optimizer with learning rate equal to `1e-03`. The batch size is set to 16 . To perform early stopping, we evaluate, after each epoch, on a small subset of each of the three validation sets. Each epoch corresponds to going through some random 10% of the training set; in addition to this, we select a constant subset of each validation set, sacrificing some accuracy in order to speed up the process.

After each epoch, validation stats are printed and a new model file is saved to the `./model` folder. We submitted a few models from different epochs. In particular, it seemed that Validation set 1 overestimated the MAPE metric on the test set, whereas validation sets 2 and 3 followed the test set's MAPE more closely. Submissions that prioritized losses on validation sets 2 and 3 were usually more successful (unless the MAPE on validation 1 was significantly large).

Training on Model 1 took just over 8 hours. Training on Model 2 takes just over an hour. The respective model weights were saved. While compiling the initial report, we loaded the model weights and confirmed that they indeed produce the same submissions sent to the challenge.

Tensorboard support was added to the code a few days after training Model 1. It provides another way to look at the performance metrics on-the-fly. The training curves for Model 2 are shown in Fig. 2.

The obtained results are listed in Table 4, listing the mean absolute percentage error for the validation subsets 1/2/3, as well as for the final test set.

Table 4 – MAPE error for the validation and test sets. We report results for our 2 model architectures, and their average. In addition, we investigated a few variations of Model 1: using the baseline prediction, not using the higher-level QT features, and not using those features or dividing the original features by `p_AvgPktsLambda`

	Val. 1	Val. 2	Val. 3	Test
Model 1	2.71	1.33	1.65	1.45
Model 2	3.61	1.17	1.55	1.45
Average of predictions	—	—	—	1.27
Baseline	12.10	9.18	9.51	10.42
M1 w/o QT	6.02	9.78	9.30	7.18
M1 w/o QT or div.	8.20	45.64	250.34	85.56

The final versions of both models performed almost identically on the test set. On the validation sets, Model 1 was better than Model 2 on Validation set 1, and worse on validation sets 2 and 3. Their average was able to attain the lowest MAPE of 1.27.

A few other models were evaluated for comparison. The analytical baseline on its own was able to achieve a respectable MAPE of 10.42 on the test set. This is slightly worse than the 7.18 MAPE of Model 1 without the QT features (`M1 w/o QT`). If we also forget to divide features by `p_AvgPktsLambda` as mentioned previously (`M1 w/o QT or div.`), the model completely fails to generalize to the

 © International Telecommunication Union, 2022

test set and validation sets 2 and 3. One thing of note is that the training curves can sometimes be unstable: for example, the curve for `M1 w/o QT` eventually rose to 100 MAPE on Validation set 3. Therefore, having a validation set (even if it consists of just a few examples) is immensely useful to check for generalization and perform early stopping.

When we don't put any measures in place to generalize to larger graphs, Validation set 1 seems least affected. The bad result of 85.56 MAPE is still better than the 300 reported by the organizer's result for the original Routenet [11]. One possibility is that summing the predictions at each link as a final step leads to better generalization than directly predicting delay on each path.

The rules of the GNNet Challenge allowed up to 20 submissions. For each submission, the error on the test set was made immediately available to the competitors. Even though we used less than 20 submissions, it is possible that this measure is (slightly) optimistic. Nonetheless, it is apparent that the introduction of higher-level features makes a huge difference when generalizing to larger graphs, and that QT-Routenet outperforms both Routenet and analytical approaches in this scenario.

5. CONCLUSION

This paper presented a novel approach to generalize per-path-delay predictions to larger 5G networks. We managed to avoid some of the limitations of graph neural networks by working directly at the input level. Namely, we used a robust baseline based on queueing theory to extract higher-level features, and then fine-tuned them to improve the baseline without sacrificing generalization. We achieved good results in the test set and all 3 validation subsets that exploited different path lengths and link capacities. The proposed solution achieved first place in the GNNet Challenge 2021 and second place overall in the ITU AI/ML in the 5G challenge. We hope that this work will help develop more approaches that fine-tune a robust approximate model that generalizes well to different distributions. For future work, we expect better results when combining our approach with other solutions developed during this challenge.

ACKNOWLEDGEMENT

This study was financed in part by the Coordenação de Aperfeiçoamento de Nível Superior - Brasil (CAPES) - Finance Code 001.

REFERENCES

[1] *ITU Challenge participation numbers*. https://aiforgood.itu.int/meet-the-winning-teams-in-the-itu-ai-ml-in-5g-challenge. Accessed: 2022-01-26.

[2] Krzysztof Rusek, José Suárez-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: *Proceedings of the 2019 ACM Symposium on SDN Research*. 2019, pp. 140–151.

[3] Suárez-Varela et al. "The Graph Neural Networking Challenge: A Worldwide Competition for Education in AI/ML for Networks". In: *SIGCOMM Comput. Commun. Rev.* 51.3 (July 2021), pp. 9–16. ISSN: 0146-4833. DOI: 10.1145/3477482.3477485. URL: https://doi.org/10.1145/3477482.3477485.

[4] Moshe Zukerman. "Introduction to queueing theory and stochastic teletraffic models". In: *arXiv preprint arXiv:1307.2968* (2013).

[5] András Varga. "The OMNET++ discrete event simulation system". In: *Proc. ESM'2001* 9 (Jan. 2001).

[6] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. "A Comprehensive Survey on Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.1 (2021), pp. 4–24. DOI: 10.1109/TNNLS.2020.2978386.

[7] Martin Happ, Jia Lei Du, Matthias Herlich, Christian Maier, Peter Dorfinger, and José Suárez-Varela. "Exploring the Limitations of Current Graph Neural Networks for Network Modeling". In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2022, pp. 1–8.

[8] Miquel Ferriol-Galmés, José Suárez-Varela, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. "Scaling Graph-based Deep Learning models to larger networks". In: *arXiv preprint arXiv:2110.01261* (2021).

[9] Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. 2019, pp. 8024–8035.

[10] Matthias Fey and Jan E. Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.

[11] *GNNet Challenge 2021 slides*. http://2ja3zj1n4vsz2sq9zh82y3wi-wpengine.netdna-ssl.com/wp-content/uploads/2020/12/GNNet_challenge_2021-2.pdf. Accessed: 2022-01-26.

[12] Bruno Klaus de Aquino Afonso and Lilian Berton. "Optimizing Diffusion Rate and Label Reliability in a Graph-Based Semi-supervised Classifier". In: *Brazilian Conference on Intelligent Systems*. Springer. 2021, pp. 514–527.

[13] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. “Structured sequence modeling with graph convolutional recurrent networks”. In: *International Conference on Neural Information Processing*. Springer. 2018, pp. 362–373.

[14] Benedek Rozemberczki et al. “PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models”. In: *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*. 2021, pp. 4564–4573.

[15] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. “Graph Attention Networks”. In: *International Conference on Learning Representations*. 2018.

[16] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. “RouteNet: Leveraging Graph Neural Networks for network modeling and optimization in SDN”. In: *IEEE Journal on Selected Areas in Communications* 38.10 (2020), pp. 2260–2270.

AUTHORS

Bruno Klaus de Aquino Afonso is a PhD candidate at the Federal University of São Paulo (ICT-UNIFESP), where he previously received his M.S. and B.S. degrees in computer science. Has been the recipient of fellowships from the Coordination for the Improvement of Higher Education Personnel (CAPES) and The São Paulo Research Foundation (FAPESP). He is interested in making the most out of labels in graph-based semi-supervised learning. While researching GNNs, he stumbled upon ITU’s challenge and the unfamiliar territory of queueing theory and traffic flow in 5G networks. With some perseverance and good fortune, he ended up as the Silver Champion of said challenge.

Lilian Berton received a B.S. degree in computer science and a licentiate degree in mathematics from Midwest University, Brazil in 2007 and 2008, respectively. She received her M.S. and Ph.D. in computer science from the University of São Paulo in 2009 and 2016, respectively. She had fellowships from FAPESP. Currently, she is an associate professor at the Institute of Science and Technology – Federal University of São Paulo, Brazil. Her research interests are machine learning, graph-based methods, complex networks and data mining.

© International Telecommunication Union, 2022

APPENDIX A - MODEL 1 CODE

Algorithm 1 Model 1 (submitted on September 22nd)

Require:

$\boldsymbol{X} = \texttt{Concatenate}([\boldsymbol{X}_P, \boldsymbol{X}_{Ph}, \boldsymbol{X}_L, \boldsymbol{X}_{Lh}, \boldsymbol{X}_N, \boldsymbol{X}_{Nh}], \texttt{axis=1})$

Require: B_path, B_link: baseline predictions
Require: $\boldsymbol{E}$: network topology
Require: NUM_iterations: # of message-passing iterations

E_lp_list $\leftarrow$ SeparateEdgeTimeSteps($\boldsymbol{E}_{LP}$)
$\boldsymbol{X} \leftarrow$ MLP_1($\boldsymbol{X}, \boldsymbol{E}_{LN}$)
for $0 \leq i <$ NUM_ITERATIONS **do**
▷ Paths receive messages
$\boldsymbol{X}_{Ph} \leftarrow \texttt{LeakyRELU}(\texttt{Conv}_{i,\texttt{node_to_path}}(\boldsymbol{X}, \boldsymbol{E}_{NP}))$
$H \leftarrow$ None
for $0 \leq k <$ E_lp_list.$length$ **do**
$H \leftarrow (\texttt{GConvGRU}_{0,\texttt{link_to_path}}(\boldsymbol{X}, H, \texttt{E_lp_list[k]}))$
end for
$\boldsymbol{X}_{Ph} \leftarrow$ LeakyRELU(H/(E_lp_list.length))
$(\boldsymbol{X}_{Ph})$[:, 0:B_path.shape[1]] $\leftarrow$ B_path
▷ Nodes receive messages
$\boldsymbol{X}_{Nh} \leftarrow \texttt{LeakyRELU}(\texttt{Conv}_{i,\texttt{path_to_node}}(\boldsymbol{X}, \boldsymbol{E}_{PN}))$
$\boldsymbol{X}_{Nh} \leftarrow \boldsymbol{X}_{Nh} + \texttt{LeakyRELU}(\texttt{Conv}_{i,\texttt{link_to_node}}(\boldsymbol{X}, \boldsymbol{E}_{LN}))$
▷ Links receive messages
$\boldsymbol{X}_{Lh} \leftarrow \texttt{LeakyRELU}(\texttt{Conv}_{i,\texttt{node_to_link}}(\boldsymbol{X}, \boldsymbol{E}_{NL}))$
$\boldsymbol{X}_{Lh} \leftarrow \texttt{LeakyRELU}(\texttt{Conv}_{i,\texttt{path_to_link}}(\boldsymbol{X}, \boldsymbol{E}_{PL}))$
$(\boldsymbol{X}_{Lh})$[:,0:B_link.shape[1]] $\leftarrow$ B_link
end for
$\boldsymbol{L} \leftarrow$ Concatenate(X_L, X_{Lh})
$\boldsymbol{L} \leftarrow$ Sigmoid(MLP_2($\boldsymbol{L}$)) ▷ Predicts average queue utilization
return GetPathDelay($\boldsymbol{L}, \boldsymbol{E}_{LP}$) ▷ Obtains per-path-delay

APPENDIX B - MODEL 2 CODE

Algorithm 2 Model 2 (submitted on September 29th)

Require: $\boldsymbol{X} = \texttt{Concatenate}([\boldsymbol{X}_P,\boldsymbol{X}_{Ph},\boldsymbol{X}_L,\boldsymbol{X}_{Lh}],\texttt{axis=1})$
Require: B_path, B_link: baseline predictions
Require: $\boldsymbol{E}$: network topology
Require: NUM_iterations: # of message-passing iterations
E_lp_list $\leftarrow$ SeparateEdgeTimeSteps($\boldsymbol{E}_{LP}$)
$\boldsymbol{X} \leftarrow$ MLP_1($\boldsymbol{X}, \boldsymbol{E}_{LN}$)
for $0 \leq i <$ NUM_ITERATIONS **do**
▷ Paths receive messages
$H \leftarrow$ None
for $0 \leq k <$ E_lp_list.$length$ **do**
$H \leftarrow (\texttt{GConvGRU}_{0,\texttt{link_to_path}}(\boldsymbol{X}, H, \texttt{E_lp_list[k]}))$
end for
$\boldsymbol{X}_{Ph} \leftarrow$ LeakyRELU(H/(E_lp_list.length))
$(\boldsymbol{X}_{Ph})$[:, 0:B_path.shape[1]] $\leftarrow$ B_path
▷ Links receive messages
$\boldsymbol{X}_{Lh} \leftarrow \texttt{LeakyRELU}(\texttt{Conv}_{i,\texttt{path_to_link}}(\boldsymbol{X}, \boldsymbol{E}_{PL}))$
$(\boldsymbol{X}_{Lh})$[:, 0:B_link.shape[1]] $\leftarrow$ B_link
end for
$\boldsymbol{L} \leftarrow$ Concatenate(X_L, X_{Lh})
$\boldsymbol{L} \leftarrow$ Sigmoid(MLP_2($\boldsymbol{L}$)) ▷ Predicts average queue utilization
return GetPathDelay($\boldsymbol{L}, \boldsymbol{E}_{LP}$) ▷ Obtains per-path-delay

APPENDIX C - BASELINE CODE

Algorithm 3 Baseline

Require: $\boldsymbol{E}$: network topology
Require: p_PktsGen: Packets generated per time unit for each path
Require: l_LinkCapacity: Vector w/ capacity of each link
Require: NUM_iterations: number of iterations
Initialize PB as a vector with constant values for each link.
$B \leftarrow 32$
queue_size $\leftarrow 32000$
Let $\lambda_{k,i}$ be the amount of traffic from path k passing through link i.
$\lambda_{k,k(i)}$ is the traffic from path k passing through its i-th edge.
for $0 \leq it <$ NUM_iterations **do**
$A \leftarrow$ p_PktsGen ▷ A is the demand on each path
for each path k **do**
Let m_k be the max number of edges in path k.
$\lambda_{k,k(1)} \leftarrow A_k$
$\forall j \in \{1.. \leq m_k\} : \lambda_{k,k(j)} \leftarrow A_k \prod_{i=1}^{j-1} PB_i$
end for
for each link l **do** ▷ Get total traffic on links
$T_l \leftarrow \sum_{\exists i:l=k(i)} \lambda_{k,l}$
$\rho_l \leftarrow T_l/\texttt{l_LinkCapacity}_l$
$\texttt{PB}_l \leftarrow \frac{(1-\rho_l)\rho_l^B}{(1-\rho_l)^{B+1}}$ ▷ Update blocking probabilities
end for
end for
$\pi_0 \leftarrow (1-\rho)/(1-\texttt{pow}(\rho, B+1))$ ▷ Prob. that the queue is at state 0
$\boldsymbol{L} \leftarrow \frac{1}{B}(\pi_0 + \sum_{j=1}^{B} j(\pi_0 \cdot \texttt{pow}(\rho, j)))$
baseline_link $\leftarrow [\pi_0, \rho, \boldsymbol{L}]$ ▷ Obs: Only $[\boldsymbol{L}]$ for Model 1
$\boldsymbol{L} \leftarrow \frac{\boldsymbol{L} \times \texttt{queue_size}}{\texttt{l_LinkCapacity}}$
baseline_path $\leftarrow$ GetPathDelay($\boldsymbol{L}, \boldsymbol{E}_{LP}$)
return baseline_link, baseline_path

© International Telecommunication Union, 2022