GRAPH-NEURAL-NETWORK-BASED DELAY ESTIMATION FOR COMMUNICATION NETWORKS WITH HETEROGENEOUS SCHEDULING POLICIES

Martin Happ[1,2], Matthias Herlich[1], Christian Maier[1], Jia Lei Du[1], Peter Dorfinger[1]
[1]Intelligent Connectivity, Salzburg Research, Austria, [2]IDA LAB, University of Salzburg, Austria

NOTE: Corresponding authors: Martin Happ, martin.happ@sbg.ac.at; Peter Dorfinger, peter.dorfinger@salzburgresearch.at

Abstract – Modeling communication networks to predict performance such as delay and jitter is important for evaluating and optimizing them. In recent years, neural networks have been used to do this, which may have advantages over existing models, for example from queueing theory. One of these neural networks is RouteNet, which is based on graph neural networks. However, it is based on simplified assumptions. One key simplification is the restriction to a single scheduling policy, which describes how packets of different flows are prioritized for transmission. In this paper we propose a solution that supports multiple scheduling policies (Strict Priority, Deficit Round Robin, Weighted Fair Queueing) and can handle mixed scheduling policies in a single communication network. Our solution is based on the RouteNet architecture as part of the "Graph Neural Network Challenge". We achieved a mean absolute percentage error under 1% with our extended model on the evaluation data set from the challenge. This takes neural-network-based delay estimation one step closer to practical use.

Keywords – Communication networks, delay estimation, graph neural networks, scheduling

1. INTRODUCTION

There has been an increasing use of machine learning techniques for various kinds of problems in recent years. Due to the variety of problems, many new machine learning algorithms have been developed. In particular for data that can be described by graphs, there has been an important new development known as "Graph Neural Networks" [1]. Examples of such data are chemical elements or communication networks. We focus on the latter in this paper. In this context, a communication network can be characterized by nodes and edges, where the edges represent the links between nodes. Additionally, there are properties associated with each node and each edge. This is the basic setting of Graph Neural Networks (GNNs). GNNs use the so-called "Message Passing" algorithm [2] and can express the notion of nodes and edges. However, for communication networks it is also important to consider paths (and network flows) along several consecutive links. RouteNet [3, 4] is an implementation of this idea that allows expressing paths. The RouteNet architecture consists mainly of two gated recurrent neural networks that are responsible for calculating path and link properties, respectively.

The RouteNet architecture can be used to predict per-flow performance metrics such as average delay and jitter. This can be useful for assessing networks with respect to different loads without needing to test them in reality. Hence, it is possible to determine if a network can handle a certain load with respect to a performance metric such as average delay. An alternative to such a prediction with neural networks is a simulation, using simulators such as OMNeT++ [5]. However, such simulations may be time-consuming. If the communication network itself or any settings are changed, the simulation has to be repeated. Thus, it becomes especially time-consuming when simulating the impact of a series of parameter changes. In contrast, the time-consuming training of neural networks has to happen only once in general. Hence, prediction with neural networks usually provides a faster way to estimate the performance of networks.

2. RELATED WORK

There are classical (i.e. non-machine learning) methods to predict delays in communication networks, like queuing theory [6], network calculus [7] and simulation-based approaches [5].

Boutaba et al. [8] provide a general overview on the application of machine learning to communication technologies and network measurements. The approaches in particular differ in whether the data used for learning comes from network simulators (e.g. from OMNeT++ as in our case) or from actual measurements. In addition, they can be divided into supervised, unsupervised and reinforcement learning. The approach considered in this paper is an example of supervised learning.

Mestres et al. [9] investigate modeling and prediction of delays in communication networks with feed-forward neural networks. They predict the latency based on the traffic configuration. In contrast to the RouteNet architecture, a neural network has to be trained for each specific communication network. Graph neural networks and message passing were first introduced by Scarselli et

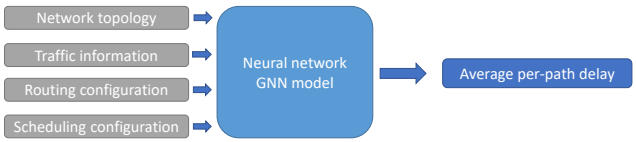

Fig. 1 – Problem setting for the challenge

al. [1] and Gilmer et al. [10]. These concepts were applied to the domain of communication technologies by Geyer and Carle [11], where the authors use a GNN for automatic network protocol design.

A different approach to deal with heterogeneous scheduling policies was recently proposed by Ferriol et al. [12]. They provide a GNN architecture with states for links, paths and queues. This model reaches a mean relative error of 3.88% in the German Backbone Network (GBN) topology (which was not used for the training process).

3. SETTING

The solution proposed in this paper was developed within the GNN Challenge [13] provided by the "Barcelona Neural Networking Center" from the Universitat Politècnica de Catalunya. This challenge was organized as part of the "ITU Artificial Intelligence/Machine Learning in 5G Challenge".

The goal of the GNN challenge was to predict the average per flow delay in a communication network. In other words, it is of interest to estimate the average time it takes for a packet to travel from its source node to its destination node. Additionally, the network topology may be different from that used in the training data. Thus, the neural network should not be adapted to only one topology but work for general topologies. For this, three different network topologies have been provided. For training, the NSFNET topology with 14 nodes [14] and GEANT2 topology with 24 nodes [15] were used. For validation, GBN [16] with 17 nodes was used. The data set that was used for the evaluation of the challenge consisted of 19 nodes. Other information has not been published by the challenge organizers. Thus, the proposed solution for this challenge must work even for such unknown communication networks where no details are known beforehand.

The data set consisted of different node and link information, as well as different settings used in the OMNeT++ simulator. A crucial simplification for all data sets is that there exists only one flow for each path. And for each flow, a Type of Service (ToS) is randomly assigned. That means all packets of a path have the same ToS. This is an important property of the data set and we will utilize it in Section 4.3. The number of generated packets per time unit follow a Poisson distribution, and the inter-packet arrival times follow an exponential distribution. A two-valued distribution is used to model packet size. The maximum bit rate is chosen randomly between 400 and 2000 bits per time unit. For more details on the simulated data, we refer to the challenge documentation [13].

In communication networks, scheduling policies describe in which order packets are transmitted. A simple and straightforward algorithm is FIFO, where the packets are transmitted in the order in which they are received [17]. The original RouteNet was developed for networks that use only a single scheduling policy. However, this implementation does not work well with other scheduling algorithms and networks with heterogeneous scheduling can have large a impact on the behavior of networks and thus on the delays. For this challenge specifically, three different scheduling policies, namely Weighted-Fair-Queuing (WFQ), Strict Priority (SP), and Deficit Round Robin (DRR) are being used. For WFQ and DRR, there are three ToS classes. The networks are in general not homogeneous with respect to scheduling policies, in fact there are data sets where all policies are present in a single communication network.

3.1 RouteNet

RouteNet uses Graph Neural Networks and the so-called message passing [2] for predicting average per-path delays in communication networks. There are two important elements in this architecture, links and paths; where each path consists of at least one link. Note that we use the term "capacity" to refer to the tight upper bound on the transmission rate of a link and the term "data rate" to refer to the desired transmission rate of a traffic flow on a network path. Each link is associated with specific information, such as link capacity. We will refer to this simply as link state information, which is represented as a vector. The same holds true for paths, which we will call analogously path state information. The RouteNet version provided for the challenge [18] uses at initialization only link capacity (bits/time unit) for the link state information h_l and the average data rate (bits/time unit) of a single flow for the path state information h_p. The data rate of the flow can be encoded as part of the path state information as in RouteNet it is assumed that there is at most one flow per path. The dimension of link and path information are both set to 32 in this RouteNet version, that is at the beginning only one component of the link and path state contains meaningful information, all other components are filled with zeros. Therefore, those two vectors can be written as

$$h_l = [x, 0, \dots, 0]' \in \mathbb{R}^{32} \text{ and}$$
$$h_p = [z, 0, \dots, 0]' \in \mathbb{R}^{32},$$

where x denotes the link capacity and z the average desired data rate on that path.

RouteNet utilizes two recurrent neural networks G_p and G_l. The neural network G_p calculates the new path state information based on the previous path information and link information. The result of this neural network is then used for G_l to calculate new link state information with the previous link state information. See Algorithm 1 for the pseudo-code. As G_p is a recurrent neural network, it returns the hidden path state after each link of a path.

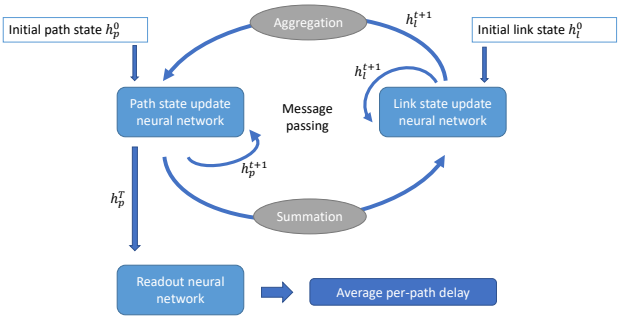

Fig. 2 – Schematic representation of RouteNet

All these states are combined to a sequence of path states for each path. Therefore, the output from G_p is aggregated for each link with a function that is denoted by f. In RouteNet, this f is equal to a summation. The function g reduces the output that is returned by G_p to the last state only, which is considered to be the new path state information. This algorithm or message passing between these two neural networks is repeated $T = 8$ times. The number of repetitions should be of the order of the average shortest path length [2]. The final path information is then an approximation of the fixed point of this message passing procedure. It is then used to predict the average delay with an additional neural network. Figure 2 gives a simplified overview of this message passing. Rusek et al. [4] give more details about the RouteNet architecture.

Data: path state h_p and link state vector h_l
Result: predicted per-path delay $\hat{y}_p$
for *t = 0 to T* **do**
$H_p^{t+1} = G_p(h_p^t, h_l^t)$
$h_l^{t+1} = G_l(f(H_p^{t+1}), h_l^t)$
$h_p^{t+1} = g(H_p^{t+1})$
end
$\hat{y}_p = R(h_p^T)$;

Algorithm 1: RouteNet architecture

4. PROPOSED SOLUTION

Our proposed solution is a modification of RouteNet [3], which is based on message passing and graph neural networks. Instead of just providing the final architecture, we give an overview of all changes we applied to the original RouteNet model and provide intermediate results for the delay predictions. That way, it is possible to see and evaluate the impact that different changes had on the results. All variants have been repeated 5 times to also assess the stability and variability of each model. Note that this number of 5 replications is arbitrary and no sample size calculation was done to compare different variants with each other given a pre-specified power for the statistical analysis. We use 600 000 training steps for each run and an exponential decay after every 60 000 steps. That means the learning rate of 0.001 is multiplied by the factor 0.6 after 60 000 training steps. Regularization is the same as in the RouteNet implementation provided for the challenge [18], that is the L^2 regularization is set to 0.1 and 0.01 for all neural networks in the first hidden layer and second hidden layer of the readout neural network. In the following we illustrate the impact of all changes on the mean absolute percentage error.

4.1 Baseline

The task was to minimize the mean absolute percentage error of per-path delays. Hence, we decided to change the loss function in the original implementation from Mean Squared Error (MSE) to Mean Absolute Percentage Error (MAPE) to use the same metric for training and evaluation.
We compare this first change with the baseline code where the optimization is done with respect to the mean squared error. The results are displayed in Table 1 as Step 0 and Step 1. It shows that without any modifications the model does not perform well as the average error over 5 runs is over 200%. This is not surprising as the original RouteNet model was developed for networks with a different scheduling policy. Using the mean absolute percentage error as the target function improved the model significantly. The grand mean of all results was about 46% (with a 95% Confidence Interval (CI) of [26.5%, 66.29%]). This improvement was expected as the results were evaluated by the mean absolute percentage error and the training was done with the same target function.

4.2 Normalization

For neural networks, it is common and advised to standardize the input variables [19, 20]. Therefore, all variables were shifted into $[0, 1]$ such that they are on the same scale. No centering was applied. This modification significantly improved the results given in Table 1. The grand mean is about 23% (95% CI [23.7%, 23.74%]). It shows again, what is already known in the literature, that normalizing or standardizing input variables is crucial and should be done. Not only to improve prediction but also to improve stability of training the model, which is reflected in a small standard deviation of those 5 runs.

4.3 Adding variables

In Step 3 we added all variables that are provided in the data set from the challenge to either path state information h_p or link state information h_l. When referring to such variables, we will provide the names of the variables as named in the data sets in parenthesis to make cross-referencing the source code easier. As the dimension is still greater than the number of variables, all unused components of h_p and h_l are again initialized with 0. To be precise, we added link capacity (`bandwidth`), the scheduling policy (`schedulingPolicy`) and weights for scheduling as link information. As there are three different ToS,

there are also three weights for the policies WFQ and DRR. For the policy SP, we artificially set these three weights to 1.

For the scheduling policy, we used dummy variables, since there are three different policies. Let $\mathbf{e}_i = (e_{i1}, e_{i2}, e_{i3})' \in \mathbb{R}^3$ be the ith canonical vector with $e_{ij} = 1$ if $i = j$ and $e_{ij} = 0$ otherwise. Note that e_i' denotes the transpose of e_i. For simplicity, the scheduling policy s is identified as integers 0, 1 or 2. Then the dummy variable for the scheduling policy can be written as e_{s+1} (sometimes known as "one hot" encoding).

For this particular data set there is exactly one flow for each path as already mentioned in Section 3. Hence, we can identify paths with flows and can therefore assign each path a ToS. That is why we can use ToS as path information. Other variables for the path information are the average data rate on that path (`AvgBw`), the generated packets (`PKtsGen`), average bit rate per time unit (`EqLambda`), average number of packets of average size generated per time unit (`AvgPktsLambda`), information about packet sizes (`AvgPktSize`, `PktSize1`, `PktSize2`) and a variable describing the upper limit for the inter-packet arrival times used in the OMNeT++ simulation (`ExpMaxFactor`). All these variables were as well shifted into $[0, 1]$ to improve the stability of the model.

We decided to split the average desired data rate on a path (`AvgBw`) into different variables for each ToS, respectively. For example, if the ToS is 1, then the first of these three variables contains the average data rate, while the other two are set to 0. For illustration, let $d \in \mathbb{R}_{\geq 0}$, $t \in \{0, 1, 2\}$ be the data rate and ToS, where the ToS is identified by integers. Then this data rate dummy variable can be written as $d \cdot \mathbf{e}_{t+1}$. We also used ToS additionally for the initial path state information. It should be noted that many of those variables listed above are highly correlated. However, we did not encounter any problems and decided to keep these variables without any further modification. By adding these additional variables, we now take into account the scheduling and therefore the prediction of average delays improved significantly.

For illustration, the state information are given by

$$h_l = [x, w_1, w_2, w_3, e'_{s+1}, 0, \dots, 0]' \in \mathbb{R}^{32} \text{ and}$$
$$h_p = [z \cdot e'_{t+1}, \dots, 0]' \in \mathbb{R}^{32},$$

where x denotes the link capacity, w_i ($i = 1, 2, 3$) for the weights, s for the scheduling policy, t for the ToS and z for the average path data rate.

Note that some variables are node properties in the data set, for example the queue scheduling policy that is used. However, flows have a direction. Let us consider a flow on the link from node A to node B. Then we assign this link the scheduling policy from the source node A. Conversely, if we have a flow in the opposite direction on the link from node B to node A, then we assign the scheduling policy from node B. Although both links connect the same nodes, they are treated as different links.

Adding these variables improves the model as scheduling information is now used as input. The average error is about 4.46% (95% CI [3.97%, 4.94%]) as can be seen under Step 3 in Table 1.

4.4 Residual connection

For the readout neural network, we used a similar idea already used in the original RouteNet model [3]. They used a residual connection for the path information to the last hidden layer of the readout neural network. This can be seen as some kind of residual neural network [21]. However, this idea is not present in the RouteNet code provided for the challenge. The readout neural network consists of two hidden layers. The output of this neural network together with the final path state information is used as input in a second neural network with one hidden layer and without any activation function (which is equivalent to a linear activation function) as the path state information can be important for estimating the average delays. The number of neurons for this layer is chosen to be equal to the dimension of the input.

The results are similar to the earlier results. The average error for Step 4 is about 4.55% (95% CI [4.38%, 4.71%]). However, the standard deviation is reduced by a factor of about $3 = 0.39/0.13$, which means the results are stabler, which can be explained by this residual neural network. There are hypotheses that such neural networks smooth the loss function and the algorithm does get stuck less often in non-optimal local minima [21][22].

To illustrate this modification, we refer to the pseudo code 2. In contrast to the unmodified code 1, the readout neural network is separated into two feed forward neural networks. The output of the first neural network with two hidden layers and "relu" activation functions is used as input for the second neural network. Note that the path state information is used in both neural networks as input.

Data: path state h_p and link state vector h_l
Result: predicted per-path delay $\hat{y}_p$
for *t = 0 to T* **do**
  $H_p^{t+1} = G_p(h_p^t, h_l^t)$
  $h_l^{t+1} = G_l(f(H_p^{t+1}), h_l^t)$
  $h_p^{t+1} = g(H_p^{t+1})$
end
$r = R_1(h_p^T)$
$\hat{y}_p = R_2(r, h_p^T)$

Algorithm 2: RouteNet architecture with modified readout neural network

4.5 Stacked gated recurrent networks

The idea of the RouteNet architecture is that for each path/flow we have information about all links of which the path consists. And this link information is used as input in a gated recurrent neural network. The initial infor-

mation is the current path state. For the first cell of the gated recurrent unit (GRU), the information of the first link of a path is being used as the input. Then, the result of this first calculation, and the information of the second link is used in the next step. This is done until all link information of a path has been used. This works well as long as no heterogeneous queuing is in the data. However, to tackle the additional complexity of queuing, we decided to use two gated recurrent networks stacked together. The idea of using stacked gated recurrent networks has already been used in a slightly different context [23], where neural networks predicted traffic volume in road networks to relieve traffic congestion. Furthermore, these gated recurrent networks seemingly allow for more flexibility as more parameters can be trained. The average error for this Step 5 is about 3.18% (95% CI [2.94%, 3.41%]); see Table 1.

Data: path state $h_{p,1}$, $h_{p,2}$ and link state vector h_l
Result: predicted per-path delay $\hat{y}_p$
for *t = 0 to T* **do**
  $H_p^{t+1} = G_p(h_{p,1}^t, h_{p,2}^t, h_l^t)$
  $h_l^{t+1} = G_l(f(H_p^{t+1}), h_l^t)$
  $h_{p,1}^t = g_1(H_p^{t+1})$
  $h_{p,2}^t = g_1(H_p^{t+1})$
end
$r = R_1(h_{p,1}^T, h_{p,2}^T)$
$\hat{y}_p = R_2(r, h_{p,1}^T, h_{p,2}^T)$

Algorithm 3: RouteNet architecture with stacked gated recurrent networks. Each GRN has its own hidden states denotes by $h_{p,i}$, $i = 1, 2$. The functions g_1 and g_2 return the final hidden state for each gated recurrent network.

4.6 Dimension path and link information

As the problem of prediction average delays in networks with scheduling is more complex than without scheduling, it may be necessary to have a higher dimension of path and link state information. The RouteNet code initially used a dimension of 32 for both. We tried increasing the dimension to 64, 128, and 256. For a dimension of 64, we observe a significant increase of the overall error over just using a dimension of 32. Doubling the dimension to 128 again reduces the prediction error significantly. However, using dimension 256 seems to increase the error, which may be a result of overfitting. For this setting, we did not try to add more regularization to avoid a possible overfit. But rather, we decided to set the dimension to 128 in the following. Another reason is computational complexity as we want to train the model in a reasonable amount of time. The results are given again in Table 1 where Step 6A represents dimension 64 with an average error of 2.02%, 6B with a dimension of 128 and an average error of 1.6% and 6C with a dimension of 256 and an error of 2.86%.

4.7 Neurons

The final path state information is obtained through the message passing [2] loop. This final information is mainly used for predicting the average delays in two steps, one neural network with two hidden layers each with 8 neurons. The other neural network does not contain an activation function and is described in Step 4.4. As we have increased the dimension of this path state information in the previous step from 32 to 128, it may be useful to increase the number of neurons in the first neural network responsible for prediction. The baseline number of neurons is 8. We increased this number to 128 and 256 and observed that there is no significant difference between 8, 128 or 256 neurons. The results for 128 and 256 neurons are given in Table 1 as Step 7A and Step 7B with an average error of 1.61% and 1.67%, respectively. As already mentioned, there is no difference to 8 neurons under Step 6B with an error of 1.6%. As the standard deviation of the results for 128 neurons (0.047) seems to be smaller than for just 8 neurons (0.061), we decided to include this change in our final solution. But as this decision is based on only 5 observations, it is not conclusive.

4.8 Decay rate

For the two final steps, we want to optimize this algorithm with respect to learning parameters. We tried two different approaches. First, we usually trained the models for 600 000 steps. For each 60 000 steps, the learning rate is decreased exponentially with a decay rate of 0.6. That is, the current learning rate r_n after n training steps is given by $r_n = 0.001 \cdot 0.6^{\lfloor n/60\,000 \rfloor}$ where $\lfloor . \rfloor$ denotes the floor function. This means that after 600 000 steps the learning rate is almost zero and no changes are observed anymore to the parameters. That is why we decided to increase the learning rate again after 600 000 steps artificially by changing the decay rate to 0.85. Then, the adjusted learning rate is given by $r_n = 0.001 \cdot 0.85^{\lfloor n/60\,000 \rfloor}$ for $n \geq 600\,000$. We refer to this approach as Step 8A and it is related to the concept of cyclical learning rates [24] where the learning rate is increasing and decreasing in a cyclical way.

We compared this approach where we change the decay rate in the beginning of the training to 0.85. We call this approach Step 8B. The former method returns an average error of about 1.47%, the latter an average error of 1.36% as can be seen in Table 1. A graphical representation of the loss functions up to 1.2 million training steps is given in Figure 3 and Figure 4. In Figure 3, there is an increase in the loss function after 600 000 steps as the learning rate was modified at that point. Note that for both loss functions the mean absolute percentage errors are shown shown on a log scale.

As no overfitting was observed we did not change the regularization and decided to keep the standard parameters from RouteNet. In our tests, method B where we changed the decay rate in the beginning performed slightly better.

Table 1 – Mean absolute percentage error (MAPE) for each modification step for five runs: Last two columns denote the average and standard deviation of each row.

	Step	Run 1	Run 2	Run 3	Run 4	Run 5	$\hat{\mu}$	$\hat{\sigma}$
0	MSE	337	120	335	102	185	216	114
1*	MAPE	26.4	64.1	43.7	36.9	60.9	46.4	16.0
2*	Normalization	23.7	23.7	23.7	23.7	23.7	23.7	0.01
3*	Variables	4.55	4.85	4.58	3.80	4.51	4.47	0.39
4*	Residual connection	4.45	4.75	4.53	4.41	4.59	4.55	0.13
5*	Stacked GRN	3.05	3.32	3.40	3.17	2.94	3.18	0.19
6A	Dimension path and link state (64)	2.03	1.94	1.97	1.86	2.28	2.02	0.16
6B*	Dimension path and link state (128)	1.68	1.63	1.52	1.58	1.57	1.60	0.06
6C	Dimension path and link state (256)	2.65	2.99	3.00	3.19	2.48	2.86	0.29
7A*	Neurons (128)	1.60	1.69	1.59	1.57	1.59	1.61	0.05
7B	Neurons (256)	1.59	1.80	1.71	1.60	1.73	1.67	0.09
8A	Decay rate (0.6/0.85)	1.42	1.61	1.37	1.42	1.52	1.47	0.10
8B*	Decay rate (0.85)	1.35	1.34	1.32	1.49	1.30	1.36	0.08

* Variant selected for final solution

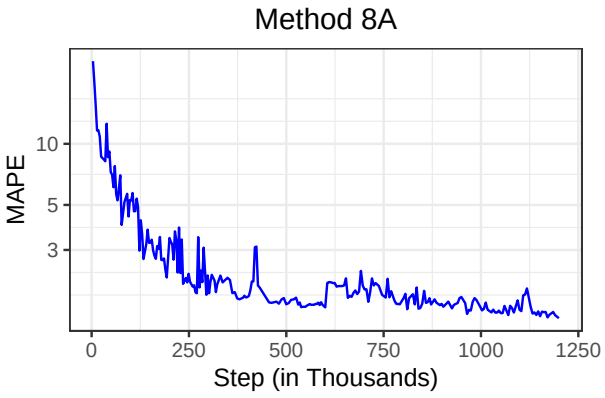

Fig. 3 – Loss function for training during Step 8A

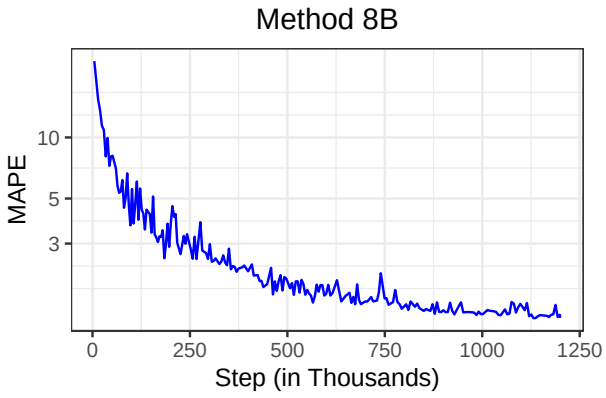

Fig. 4 – Loss function for training during Step 8B

4.9 Final results

Overall, if we evaluate the best trained model, that is run 5 from method 8B as previously described, we get a mean absolute percentage error of about 0.897% with the final data set from the GNN challenge. For comparison, the best result in the challenge by the winning team was an error of 1.53%. Our originally submitted solution achieved an error of about 1.9%, thus we could improve our submitted model further. The training was done on a single Geforce RTX 2080 Ti in under 48 hours for 1.2 million training steps. The code was written in Python 3.7.7 with tensorflow 2.1.0 based on Keras and is available as open source.[1]

5. CONCLUSION

In this paper we have described the problem of estimating delays in communication networks using deep neural networks and proposed a solution based on the RouteNet model [3]. We decomposed our solution into several steps to demonstrate the improvement of each step and compared different variants of the steps to find good hyperparameters. Such a step by step analysis of changes can be helpful in constructing, improving and understanding a model. Using this approach we were able to obtain an error of about 0.897% for predicting average per-path delays based on graph neural networks.

ACKNOWLEDGMENTS

The research was supported by the WISS 2025 (Science and Innovation Strategy Salzburg 2025) project "IDALab Salzburg" (20204-WISS/225/197-2019 and 20102-F1901166-KZP) and the 5G-AI-MLab by the Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK) and the Austrian state Salzburg.

[1] `https://github.com/ITU-AI-ML-in-5G-Challenge/GNN_Challenge_SalzburgResearch_Follow_Up_Paper`

REFERENCES

[1] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1 (2008), pp. 61–80.

[2] Krzysztof Rusek and Piotr Chołda. "Message-passing Neural Networks Learn Little's Law". In: *IEEE Communications Letters* 23.2 (2018), pp. 274–277.

[3] 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 ACM Symposium on SDN Research*. 2019, pp. 140–151.

[4] 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.

[5] András Varga and Rudolf Hornig. "An Overview of the OMNeT++ Simulation Environment". In: *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*. 2008, p. 60.

[6] Robert B Cooper. "Queueing Theory". In: *Proceedings of the ACM'81 Conference*. 1981, pp. 119–122.

[7] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Vol. 2050. Springer Science & Business Media, 2001.

[8] Raouf Boutaba, Mohammad A Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M Caicedo. "A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities". In: *Journal of Internet Services and Applications* 9.1 (2018), pp. 1–99.

[9] Albert Mestres, Eduard Alarcón, Yusheng Ji, and Albert Cabellos-Aparicio. "Understanding the Modeling of Computer Network Delays Using Neural Networks". In: *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*. 2018, pp. 46–52.

[10] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. "Neural Message Passing for Quantum Chemistry". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1263–1272.

[11] Fabien Geyer and Georg Carle. "Learning and Generating Distributed Routing Protocols using Graph-Based Deep Learning". In: *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*. 2018, pp. 40–45.

[12] Miquel Ferriol-Galmés, José Suárez-Varela, Pere Barlet-Ros, and Albert Cabellos-Aparicio. "Applying Graph-Based Deep Learning to Realistic Network Scenarios". In: *arXiv preprint arXiv:2010.06686* (2020).

[13] Barcelona Neural Networking Center. *GNN Challenge Website*. `https://bnn.upc.edu/challenge2020/`. Accessed: 2021-01-22.

[14] Xiaojun Hei, Jun Zhang, Brahim Bensaou, and Chi-Chung Cheung. "Wavelength Converter Placement in Least-Load-Routing-Based Optical Networks Using Genetic Algorithms". In: *Journal of Optical Networking* 3.5 (2004), pp. 363–378.

[15] Fernando Barreto, Emílio CG Wille, and Luiz Nacamura Jr. "Fast Emergency Paths Schema to Overcome Transient Link Failures in OSPF Routing". In: *arXiv preprint arXiv:1204.2465* (2012).

[16] João Pedro, João Santos, and João Pires. "Performance Evaluation of Integrated OTN/DWDM Networks with Single-Stage Multiplexing of Optical Channel Data Units". In: *13th International Conference on Transparent Optical Networks*. IEEE. 2011, pp. 1–4.

[17] J. Kurose and K. Ross. *Computer Networking: A Top-Down Approach, 7th Edition*. Pearson Education Limited, 2017. ISBN: 978-0-13-359414-0.

[18] Knowledge-Defined Networking. *RouteNet Challenge Github Repository*. `https://github.com/knowledgedefinednetworking/RouteNet-challenge`. Accessed: 2021-02-03.

[19] Jorge Sola and Joaquin Sevilla. "Importance of Input Data Normalization for the Application of Neural Networks to Complex Industrial Problems". In: *IEEE Transactions on Nuclear Science* 44.3 (1997), pp. 1464–1468.

[20] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. "Efficient Backprop". In: *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 9–48.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Identity Mappings in Deep Residual Networks". In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[23] Peng Sun, Azzedine Boukerche, and Yanjie Tao. "SSGRU: A Novel Hybrid Stacked GRU-Based Traffic Volume Prediction Approach in a Road Network". In: *Computer Communications* 160 (2020), pp. 502–511.

[24] Leslie N Smith. "Cyclical Learning Rates for Training Neural Networks". In: *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE. 2017, pp. 464–472.

AUTHORS

Martin Happ received his Ph.D. in Mathematics at the University of Salzburg in 2020. His research interests include multivariate statistics, especially repeated measurements, and nonparametric statistics with a focus on rank procedures. Recently, he started working on machine learning with application in communication networks in his current position at Salzburg Research.

Matthias Herlich received a doctorate degree in Computer Science from the Universität Paderborn in Germany. After working at the Palo Alto Research Center and the National Institute of Informatics in Tokyo he went to Salzburg Research. His experience ranges from analytic and simulation-based approaches to evaluation of measurements. Recently, he has begun to apply his knowledge to machine learning for communication networks.

Christian Maier received a BSc in Mathematics from the Technical University of Munich. He is now a researcher at Salzburg Research. His work focuses on reliability and bandwidth measurements in wireless networks, and on the application of machine learning to communication technologies.

Jia Lei Du received his doctorate degree (Dr.-Ing.) in Electrical Engineering from the Heinz Nixdorf Institute, University of Paderborn, Germany for the study of mobile robot cooperation and communication systems, and his diploma degree (Dipl.-Ing.) in Engineering Cybernetics from the University of Stuttgart, Germany. Jia Lei Du also worked as a researcher at Philips Medical Systems, Boeblingen, Germany where he was responsible for a key component for the reliable communication in a new patient monitoring product. He is now a researcher at Salzburg Research and his interests include communication networks, distributed systems, and multi-robot systems.

Peter Dorfinger received his diploma degree (DI(FH)) from the Department of Telecommunications Engineering at the Salzburg University of Applied Sciences in 2002. In 2010 he received his master's degree (DI) from the Department of Information Technologies and Systems Management at the Salzburg University of Applied Sciences. He is head of group, senior researcher and project manager within Salzburg Research. He has published more than 40 research papers. His research focus is on wired and wireless critical infrastructure networks with a specific focus on measurement and monitoring. Since 2007, he lectures on network protocols and services at the Salzburg University of Applied Sciences.