# SOFGNN Report

SOFGNN Team
Charles Boudreau[1], Huy Duong[2], Brigitte Jaumard[1,2], Junior Momo Ziazet[1]
[1] *Computer Science and Software Engineering, Concordia University, Montreal, Canada*
[2] *CRIM - Computer Research Institute of Montreal, Montreal, Canada*

November 29, 2021

## 1 Motivation and Problem Statement

The purpose of this project entails the creation of a "digital twin", a software approximation of a real-world object or system, used to simulate the behaviour of the original object or system under a given set of conditions. Specifically, the task calls for the elaboration of a digital twin for a network which, given inputs detailing the network's topology and traffic flows, outputs performance metrics such as delay, jitter or packet loss, among others.

Indeed, 5G and B5G networks have critical requirements in terms of performance from both a network and a user perspective. They offer service requests with very precise categorization, in order to allow network operators to design and operate dedicated slices for each service, with respect to both QoS and QoE as perceived by the users [3]. In particular, as 5G/B5G networks are expected to be highly dynamic, the challenge is to predict accurately the KPI parameters in order to ensure their requirements in terms of service assurance [8].

For this task, one important constraining factor to consider is that the proposed solution should be scalable to larger topologies (moving to 30 to 300 nodes for the objective of the GNN competition), meaning that the solution should remain effective when applied to topologies whose size greatly exceeds that of topologies that were encountered by the solution during the training process.

As the data in this problem can be structured as a graph, one possible tool that can be used to solve it is the Graph Neural Network or GNN. A Graph Neural Network is a type of deep learning architecture that takes graph data as inputs and produces node-level embeddings that encode the surrounding graph context for the nodes, which can then be used to infer properties about the individual nodes or the entire graph. GNNs can be thought of as a generalization of the Convolutional Neural Nets to non-euclidean spaces, hoping to do for graphs what Convolutional Neural Networks did for image data [1].

Graph Neural Networks [6] [7] are a good candidate solution for this problem as they demonstrate the ability to generalize to networks that were not seen during training. However, GNNs still have issues generalizing to networks whose scale is much larger than what the network has seen during training. This challenge therefore aims to address this limitation by asking participants to develop efficient and scalable GNN models.

## 2 Our Contribution

Working with an implementation of a Graph Neural Network (GNN) specifically designed to tackle this problem as a baseline, known as RouteNet [5], we set out to improve its ability to estimate the delay on topologies that are larger than those seen during training. We noticed that the main difference between training and real-world deployment was that, in the real world, we observe higher link capacity and larger network diameter which means a difference in distribution between training data and test data. We therefore worked on designing scale independent features and output to solve the out of distribution issue. Hence, we first propose a new feature that we call 'link load' which indicates relatively how a link is occupied at a particular time. It is computed as the ratio of the summation of the flow traversing the link by the link's capacity, or maximum allowable bandwidth. The goal here is to provide dynamic information that matches exactly with what is happening in the network rather than just using a static, absolute value like the link capacity as it was done in the baseline RouteNet model. This feature was later used to further improve our solution by designing two other derived features. The first is 'link load squared' (link load rased to the power two) and the second is 'link load cubed'(link load rased to the power three). This was done in order to not only help GNNs to better extrapolate by manually adding non-linearities to our features [9], but also because by doing so, less busy links would be more easily differentiated from busier ones.

The second change of significance we implemented pertains to the model's output. Originally, the RouteNet model attempts to directly predict the delay for the individual flows in the network. One issue with this approach is that the distribution of delay values in the training set differs significantly from the distribution found in the validation and test sets. In order to address this, we have the model instead predict the occupancy ratio of the outgoing queues, which has a range of possible values bound between 0 and 1, and then use these predicted values to estimate the delay using a post processing step. This change makes RouteNet much more robust to changes in input topology size.

In its unmodified state, Routenet's delay estimation on topologies of 50 to 300 nodes, given topologies of 25 to 50 nodes as training, achieves a Mean Absolute Percentage Error (MAPE) score in the range of about 300. With our modifications, the new solution is able to improve the MAPE score to under 1.5.

## 3 Background on GNN and RouteNet

A graph can be defined as a data structure consisting of two components: nodes, which represent a specific object or datapoint, and edges, which represent relations between the objects. A Graph Neural Network is a type of Neural Network that can be directly applied to graph-structured data. A GNN takes graph data as inputs and produces node-level embeddings that encode the surrounding graph context for the nodes, which can then be used to infer properties about the individual nodes or the entire graph. There are many different variations of GNNs, but at their core, most GNNs have these two basic operations: a message-passing scheme and an update function. The message-passing scheme determines how information about the nodes' state in communicated to its neighbors, and the update function determines how a node's state is updated using the messages aggregated from neighboring nodes.

RouteNet [5] is a network model based on a Graph Neural Network (GNN) that is able to understand the relationship between topology, routing, and input traffic to produce estimates of some KPI in communication networks (per-flow mean delay for our use case). The RouteNet model shows promising properties but it has some difficulty generalizing to larger graphs. The organizers of the competition proposed to consider RouteNet as a baseline. The original RouteNet model gave a Mean Percentage Absolute Error of about 300% on this year challenge task.

## 4 Solution Description

Our proposed solution is a modification of RouteNet [5], it is based on the combination of the solutions of two models, both based on the tensorflow implementation of Routenet . Sections 4.1, 4.2 and 4.3 describe in detail our contributions.

### 4.1 Output Design : Direct Delay Prediction vs Queue Occupancy Prediction

First, instead of predicting directly the delay, we predict the queue occupancy ratio of the outgoing queues. This way, the scalability issue is indirectly partially solved since instead of a path level prediction where path length can significantly differs from small size network topology to larger size network, we predict a link level measure which is not affected by the network size. By the scalability issue, we mean that the accuracy with path level prediction gets worsen when testing with larger topologies than ones seen in the training data.

Figure 1 shows a high level view of the idea where can see that, in this context, to get an accurate delay prediction, it is easier to make the summation of the queue delay and the propagation delay rather than estimating directly the delay. This solves the issue we observed which is the distribution of delay values in the training set differs significantly from the distribution found in the validation and test sets.

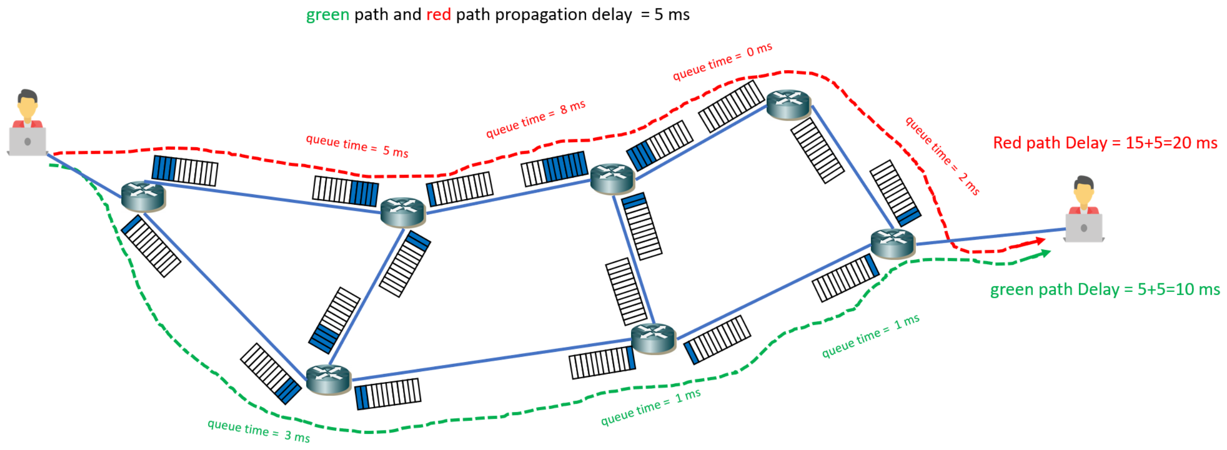


Figure 1: Total Delay as queue and propagation delay

Thus, the proposed solution predicts, for each link, the "queue occupancy", which is expressed as the average port occupancy (service and waiting queue) of the QoS queue (packets) divided by the queue size (packets). In other words, this is a normalization of the average port occupancy. We then compute the combined queue and propagation delay to obtain a delay value for a given link. Particularly, path delays can be estimated as the linear combination of delays encountered in the queues of the outgoing links of the nodes along the path, considering the average queue occupancy, the queue size, the packet size, and the capacity of the outgoing link of the queue. The link delay and flow delay formulations are represented below in Eq. (1) and Eq. (2) respectively:

$$\text{Delay}_\ell = \text{Queue\_occupancy}_\ell \times \text{Queue\_size}_\ell \times \text{Avg\_packet\_size}_\ell / C_\ell, \tag{1}$$

$$\text{Delay}_f = \sum_{k=1}^{N_f} \text{Delay}_\ell, \tag{2}$$

where $N_f$ is the number of links that defines the flow path $f$, and $C_\ell$ the capacity of link $\ell$.

## 4.2 Feature Engineering: Capacity vs Link Load

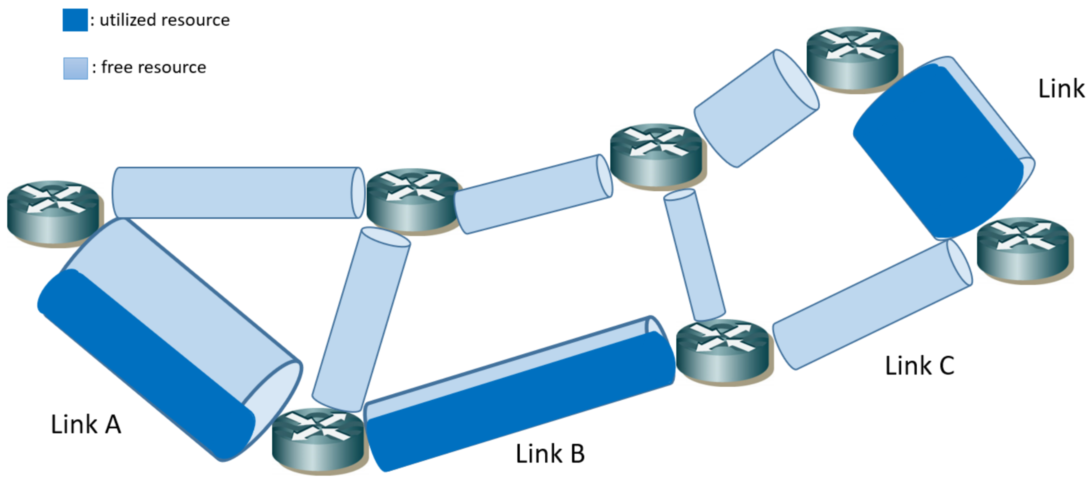
We designed a new feature that we called 'load load' which indicates relatively how a link is occupied at a particular time. The idea here was to provide to the model a feature that consistently evolve with time and that stays in the same range ([0,1]) even if we change some network parameters like the link capacity. We made that feature to replace the link capacity feature that was used in the original RouteNet model.

Eq. (3) shows the Link load formula which is expressed as the summation of the bandwidth of all flows traversing the link, divided by the link's capacity:

$$Link\_Load = \frac{\sum\limits_{f \in N_\ell} t_f}{C_\ell}, \tag{3}$$

where $t_f$ is the value of the traffic feature of flow $f$, $\mathcal{N}_\ell$ is the set of flows that traverse link $\ell$, and $C_\ell$ indicates the transport capacity of link $\ell$ (link bandwidth (bits / time unit))

Figure 2 presents how the link load changes according to the occupation of a link while capacity just stays constant.



| | Capacity | Link Load |
|---|---|---|
| Link A | 100 | 20/100 |
| Link B | 30 | 20/30 |
| Link C | 20 | 0/20 |
| Link D | 100 | 90/100 |

Figure 2: Capacity vs Link Load representation

According to Xu *et al.* [9], Graph Neural Networks as well as fully-connected neural networks using ReLU activation can only fit linear functions outside of the training support. Encoding non-linearities either through the architecture of the model or, as in this case, through the features may result in the model only needing to fit a linear function and thus allowing it to generalize to unseen data more effectively.

Based on that, we further improve our model by designing two more derived features by raising link load to the power 2 and 3 respectively called 'Link Load squared' and 'Link Load cubed'. This allowed us to add some non linearities to the input that will help the GNN extrapolate. Futhermore, raising link load to power 2 and 3 would also provide features that show significant differentiation between links that are heavily utilised from those that are not. Figure 3 illustrate and example of such differentiation.

$$Link\_Load\_Squared = \left( \frac{\sum_{f \in N_l} t_f}{C_l} \right)^2 \tag{4}$$

$$Link\_Load\_cubed = \left( \frac{\sum_{f \in N_l} t_f}{C_l} \right)^3 \tag{5}$$

### 4.3 Proposed model architecture

Figure 4 shows the overall architecture of our solution. It is an ensemble of two RouteNet based models (GNN1 and GNN2). From the orignial RouteNet model, we modified the size of the link hidden state and path hidden state as well as the number of units of each DNN to adapt it to our designed features and output. We also increased the number of message passing iteration to 12 so that is can be approximately in order of magnitude of the average shortest path length in real world networks as suggested in [4]. In our model,

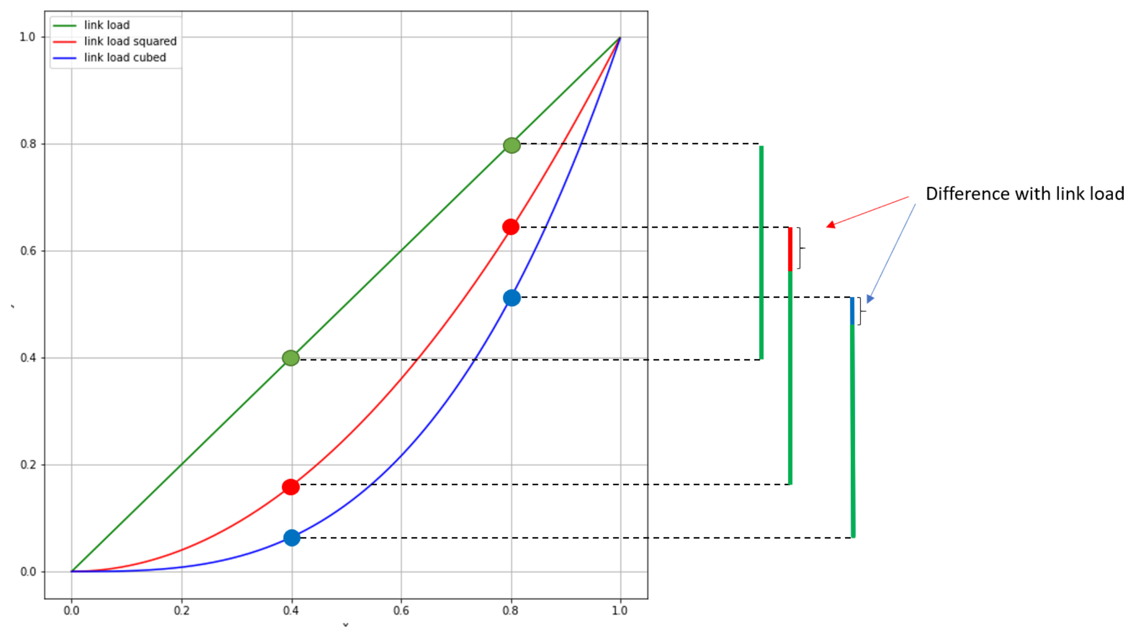


Figure 3: Effect of powering the load link

GNN1 divers with GNN2 just according to their respective inputs. While GNN1 takes as input link features our designed link load and link load squared, GNN2 has an additional feature with is link load cubed. With both models, after 12 rounds of message passing, the link hidden state will be fed to the readout function represented here with a neural network to predict the link queue occupancy. Then with a post processing step represented in Equations (1) and (2), both models will output their predictions and the final prediction will be their average.

The following input features are used in our solution:

- **Link features:**
  - Link load (see Equation (3)),
  - Link load squared ( see Equation (4)),
  - Link load cubed (see Equation (5), this feature is only used in the second model GNN2)
- **Path features:**
  - Traffic: Average bandwidth of a given traffic flow (bits/time unit)[2]
  - Packets: Packets generated by a traffic flow per time unit (packets/time unit), see [2]
  - Eqlambda: Time Distribution Feature. Average bitrate per time unit (bits/time unit), see [2]

Table 1 presents the input of each model.

## 5 Training

The model was trained on an NVIDIA GeForce RTX 2080 Ti. The hyper-parameters which were used for the proposed solution with both models are described in Table 2.

No changes were made to the message passing scheme beyond modifying the number of message passing rounds.

Table 3 presents in detail the parameters of our proposed graph neural network architecture with fully connected layers and Gate Recurrent Unit.

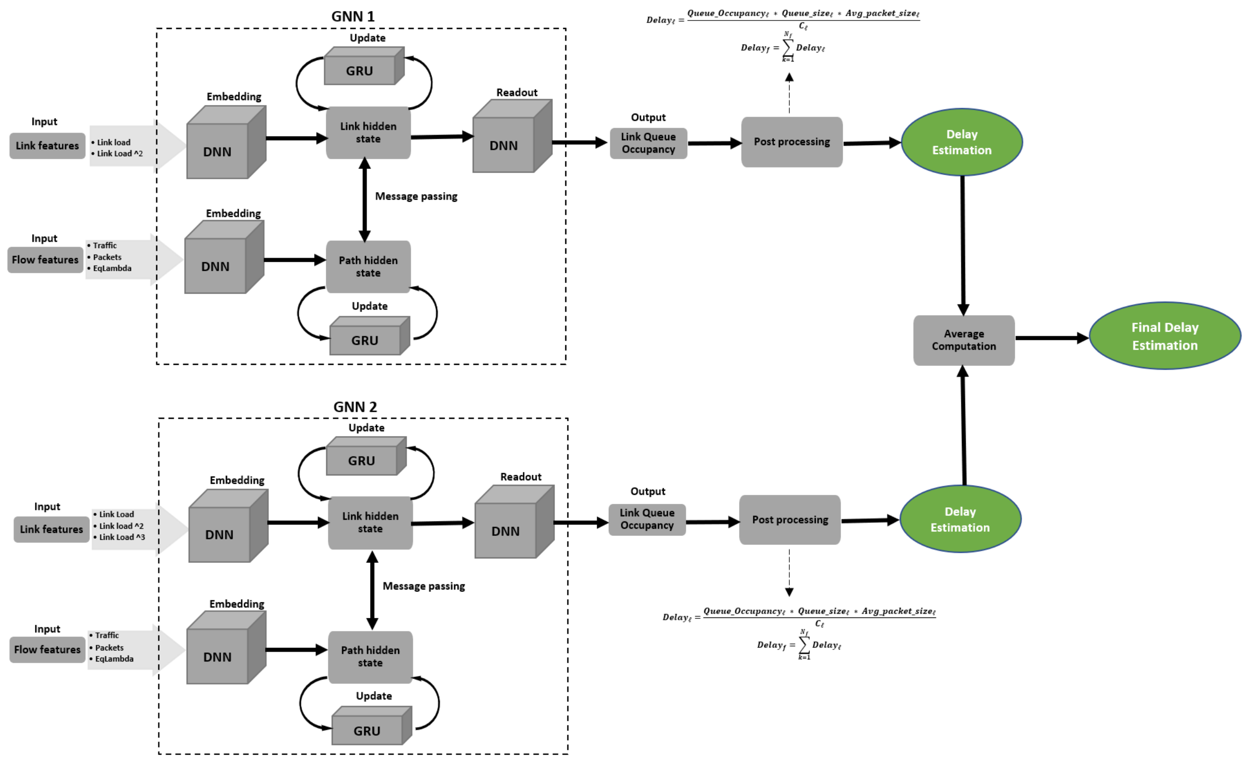


Figure 4: Architecture of our proposed solution

Table 1: Model Input Features

| | GNN 1 | GNN 2 |
|---|---|---|
| Flow features | Traffic | traffic |
| | Packets | Packets |
| | EqLambda | EqLambda |
| Link features | Link Load | Link Load |
| | Link Load squared | Link Load squared |
| | - | Link Load cubed |

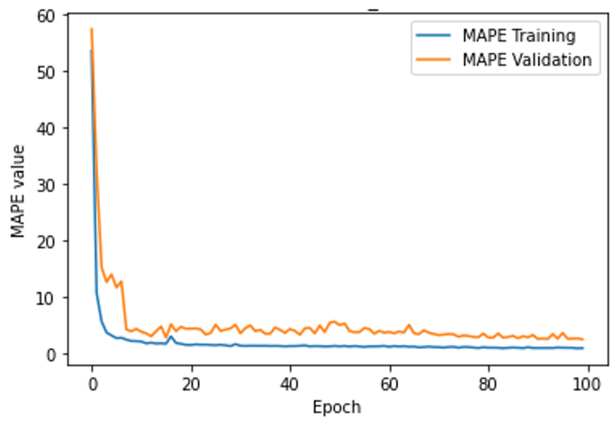
The training curves of GNN1 And GNN2 are represented in Figure 5. We can see how fast our model can learn. With just 20 epochs, our model is already under a MAPE of 5%. This is an indicator of how practical our solution is.

Table 2: SOFGNN Hyperparameters.

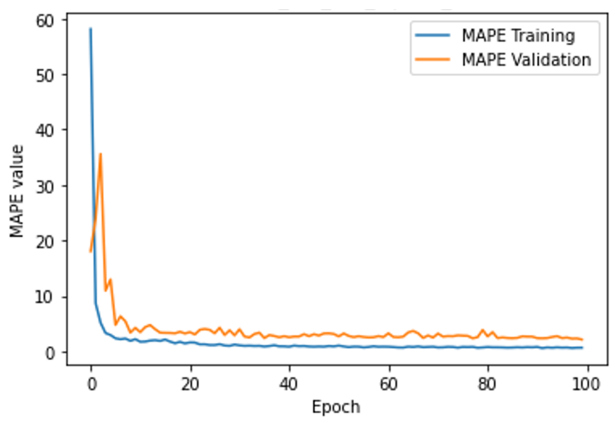
| | GNN1 | GNN2 |
|---|---|---|
| Link Hidden State Size | 128 | 128 |
| Path Hidden State Size | 128 | 128 |
| Readout Layer Size | 128 | 128 |
| Message Passing Rounds | 12 | 12 |
| Learning Rate | 0.001 | 0.001 |
| Optimizer | Adam | Adam |
| Loss | MAPE | MAPE |

Table 3: Graph Neural Networks Architectures

| Layer | Type | Activation | Number of Units |
|---|---|---|---|
| Link Embedding | | | |
| 1 | Fully conected | RELU | 128 |
| 2 | Fully conected | SELU | 128 |
| Path Embedding | | | |
| 1 | Fully conected | RELU | 128 |
| 2 | Fully conected | SELU | 128 |
| Link Update | | | |
| 1 | GRU | - | 128 |
| Path Update | | | |
| 1 | GRU | - | 128 |
| Readout | | | |
| 1 | Fully conected | RELU | 128 |
| 2 | Fully conected | RELU | 128 |
| 3 | Fully conected | None | 1 |



(a) GNN 1



(b) GNN 2

Figure 5: Training evolution

## 6 Results

We next present the numerical results obtained with our improvement for both the model and the algorithms.

Table 4 Shows the progression of our solution over the course of the challenge, detailing the type of change we made to the model and the effect on the score. As stated above, the biggest contributions to the

Table 4: MAPE for each modification to get the final solution

| | | Models | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Updates | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | Data Augmentation | | x | x | | | | | |
| 2 | Link Load | | | x | x | x | x | x | x |
| 3 | Link Load Squared | | | | | | x | x | x |
| 4 | Link Load Cubed | | | | | | | x | x |
| 5 | Direct Delay Prediction | x | x | x | | | | | |
| 6 | Occupancy Prediction | | | | x | x | x | x | x |
| 7 | Average Packet Size: per Distribution | | | | x | x | | | |
| 8 | Average Packet Size: per Flow | | | | | | x | x | x |
| 9 | Hyperparameter Tuning | | | | | | x | x | x |
| 10 | Ensemble | | | | | | | | x |
| | **MAPE(%) (validation set)** | **$\sim$300** | **$\sim$50** | **$\sim$45** | **$\sim$5** | **$\sim$2** | **$\sim$1.5** | **$\sim$1.4** | **$\sim$1.3** |

improvement of the score come from the replacement of the capacity feature with the link load feature and the switch from directly predicting the delay to estimating the delay using the predicted queue occupancy value. With the final ensemble model, we got a MAPE of 1.38% on the test data set.

## 7 Conclusion

Our results show that, while RouteNet on its own is indeed capable of scaling up return fairly accurate results for topologies larger than those it was trained on, performance is very dependent on the inputs given to it, as well as the output it is expected to return. RouteNet benefits from having the raw input data transformed in a way to help it extract relevant information from them. Addressing the difference in distribution between the training and testing sets is also of paramount importance. With these two changes, as well as other minor modifications and hyperparameter tuning, we were able to create a model that was capable remaining accurate in its per-flow delay predictions, with a final MAPE score under 2.0, even as the size of the input topology increased far beyond anything seen during training.

Future directions could involve inspecting the model's performance on even larger topologies, in order to examine how the model's performance degrades as topology size increases. Another direction to consider would be to fine-tune the model to predict other KPIs beyond per-flow delay, such as Jitter or Packet Loss, subject to various traffic loads and distributions, according to the KPI requirements of the service requests in various dedicated slices (URLLC, eMBB and mMTC). Indeed, note that very few studies exist with different traffic distributions, i.e., with chaning traffic patterns over the days or weeks.

## References

[1] M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[2] DataNet API Documentation. `https://github.com/BNN-UPC/datanetAPI/tree/challenge2021`.

[3] J.E. Preciado-Velasco, J.D. Gonzalez-Franco, C.E. Anias-Calderon, J.I. Nieto-Hipolito, and R. Rivera-Rodriguez. 5G/B5G service classification using supervised learning. *Applied Sciences*, 11(4942), 2021.

[4] K. Rusek and P. Chołda. Message-passing neural networks learn little's law. *IEEE Communications Letters*, 23(2):274–277, 2018.

[5] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio. RouteNet: Leveraging graph neural networks for network modeling and optimizationin SDN. *IEEE Journal on Selected Areas in Communications*, 38(10), 2020.

[6] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61 – 80, 2009.

[7] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4 – 24, January 2021.

[8] M. Xie, F. Michelinakis, T. Dreibholz, J.S. Pujol-Roig, S. Malacarne, S. Majumdar, W.Y. Poe, and A.M. Elmokashfi. An exposed closed-loop model for customer-driven service assurance automation. In *Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pages 419 – 424, 2021.

[9] K. Xu, M. Zhang, J. Li, S.S. Du, K. Kawarabayashi, and S. Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. *arXiv preprint arXiv:2009.11848*, 2020.