Technical Report: RouteNet-Fermi

Miquel Ferriol-Galmés, José Suárez-Varela, Jordi Paillisé-Vilanova, Pere Barlet-Ros, Albert Cabellos-Aparicio
Barcelona Neural Networking Center, Universitat Politècnica de Catalunya, Spain
Email: {miquel.ferriol, jose.suarez-varela, jordi.paillisse, pere.barlet, alberto.cabellos}@upc.edu

I. INTRODUCTION

This technical report is intended for participants of the Graph Neural Networking challenge 2022 (https://bnn.upc.edu/challenge/gnnet2022). We present RouteNet-Fermi, a custom Graph Neural Network (GNN) for network performance evaluation. This model has a network state description as input, and it produces estimates of flow-level performance metrics as output, such as delay, jitter, or loss (see Fig. 1).

We have tested the capability of this GNN model to scale to samples of larger networks than those seen during training. To this end, we have trained RouteNet-Fermi with a large dataset with thousands of samples of networks up to 10 nodes. After training, we have validated that the model produces accurate per-flow delay estimates on the validation dataset (Mean Relative Error < 5%). The validation dataset includes samples from networks up to 300 nodes.

II. ROUTENET-FERMI

This section describes the internal GNN architecture of RouteNet-Fermi (hereafter, RouteNet-F). This GNN-based model implements a custom three-stage message passing algorithm that represents key elements for network modeling (e.g. topology, queues, traffic flows). RouteNet-F supports a wide variety of features present in real-world networks, such as multi-queue QoS scheduling policies, or complex traffic models.

Figure 1 shows a black-box representation of RouteNet-F. The input of this model is a network sample, defined by: a network topology, a routing scheme (flow-level), a queuing configuration (interface-level), and a set of traffic flows characterized by some parameters. As output, the model produces estimates of relevant performance metrics at a flow-level granularity (e.g., delay, jitter, loss). Participants of the Graph Neural Networking challenge 2022 will use this model to predict the mean per-packet delay on source-destination flows.

RouteNet-F is based on two main design principles: (i) *finding a good representation* of the network components supported by the model (e.g., traffic models, routing, queue scheduling), and (ii) *exploit scale-independent features* of networks to accurately scale to larger networks unseen during training. These two aspects are further discussed in the two next subsections.

A. Representing network components and their relationships

First, let us define a network as a set of source-destination flows $\mathcal{F} = \{f_i : i \in (1, ..., n_f)\}$, a set of queues on $\mathcal{Q} = \{q_j : j \in (1, ..., n_q)\}$, and a set of links

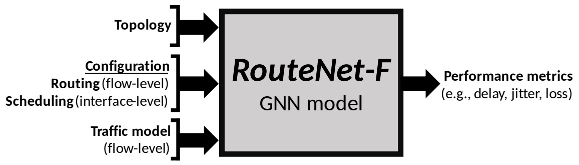

Figure 1. Black-box representation of RouteNet-F.

$\mathcal{L} = \{l_k : k \in (1, ..., n_l)\}$. According to the routing configuration, flows follow a source-destination path. Hence, we define flows as sequences of tuples with the queues and links they traverse $f_i = \{(q^{i,1}, l^{i,1}), ..., (q^{i,M}, l^{i,M})\}$, where M is the path length of the flow (number of links). Let us also define $Q_f(q_j)$ and $L_f(l_k)$ as functions that respectively return all the flows passing through a queue q_j or a link l_k. Also, $L_q(l_k)$ is defined as a function that returns the queues $q_{l_k} \in \mathcal{Q}$ injecting traffic into link l_k – i.e., the queues at the output port to which the link is connected.

Following the previous notation, RouteNet-F considers an input graph with three main components: (i) the physical links $\mathcal{L}$ that shape the network topology, (ii) the queues $\mathcal{Q}$ at each output port of network devices, and (iii) the active flows $\mathcal{F}$ in the network, which follow some specific src-dst paths (i.e., sequences of queues and links). Traffic in flows is generated from a given traffic model. From this, we can extract three basic principles:

1) The state of flows (e.g., delay, throughput, loss) is affected by the state of the queues and links they traverse (e.g., queue/link utilization).
2) The state of queues (e.g., occupation) depends on the state of the flows passing through them (e.g., traffic volume, burstiness).
3) The state of links (e.g., utilization) depends on the states of the queues that can potentially inject traffic into the link, and the queue scheduling policy applied over these queues (e.g., Strict Priority, Weighted Fair Queuing).

Formally, these principles can be formulated as follows:

$$\boldsymbol{h}_{f_i} = G_f(\boldsymbol{h}_{q^{i,1}}, \boldsymbol{h}_{l^{i,1}}, ..., \boldsymbol{h}_{q^{i,M}}, \boldsymbol{h}_{l^{i,M}}) \quad (1)$$

$$\boldsymbol{h}_{q_j} = G_q(\boldsymbol{h}_{f_1}, ..., \boldsymbol{h}_{f_I}), \quad f_i \in Q_f(q_j) \quad (2)$$

$$\boldsymbol{h}_{l_k} = G_l(\boldsymbol{h}_{q_1}, ..., \boldsymbol{h}_{q_J}), \quad q_j \in L_q(l_j) \quad (3)$$

Where G_f, G_q and G_l are some unknown functions, and $\boldsymbol{h}_f$, $\boldsymbol{h}_q$ and $\boldsymbol{h}_l$ are latent variables that encode information about the state of flows $\mathcal{F}$, queues $\mathcal{Q}$, and links $\mathcal{L}$ respectively. Note that these principles define a circular dependency between the three network components ($\mathcal{F}$, $\mathcal{Q}$, and $\mathcal{L}$) that must be solved to find latent representations satisfying the equations above.

To solve the circular dependencies defined in Equations (1)-(3), RouteNet-F implements a three-stage message passing algorithm that combines the states of flows $\mathcal{F}$, queues $\mathcal{Q}$, and links $\mathcal{L}$, and updates them iteratively. Finally, it combines these states to estimate flow-level delays. Algorithm 1 describes the architecture of RouteNet-F.

First, hidden states $\boldsymbol{h}_f$, $\boldsymbol{h}_q$, and $\boldsymbol{h}_l$ are initialized using the functions HS_f, HS_q, and HS_l respectively (lines 1-3). These functions encode the initial features $\boldsymbol{x}_f$, $\boldsymbol{x}_q$, and $\boldsymbol{x}_l$ into fixed-size vectors that represent feature embeddings. The initial features of flows $\boldsymbol{x}_f$ are defined as an n-element vector that characterizes the flow's traffic. For example, this vector includes the average traffic volume transmitted in the flow λ, and some specific parameters of the traffic model, such as t_{on} and t_{off} for On-Off traffic distributions. We set the initial features of links $\boldsymbol{x}_l$ as: (i) the link load $x_{l_{load}}$, and (ii) the scheduling policy at the output port of the link (FIFO, Strict Priority, Weighted Fair Queuing, or Deficit Round Robin). For the scheduling policy, we use one-hot encoding. The calculation of the link load $x_{l_{load}}$ is defined in more detail later (Sec. II-B1). Lastly, the initial features of queues $\boldsymbol{x}_q$ include: (i) the buffer size, (ii) the queue order/priority level (one-hot encoding), and (iii) the weight (only for Weighted Fair Queuing or Deficit Round Robin configurations).

Once all the hidden states are initialized, the message passing phase starts. This phase is executed for T iterations (loop from line 4), where T is a configurable parameter of the model. Each message passing iteration is divided into three different stages, which represent respectively the message exchanges and updates of the hidden states of flows $\boldsymbol{h}_f$ (lines 5-10), queues $\boldsymbol{h}_q$ (lines 11-14), and links $\boldsymbol{h}_l$ (lines 15-19).

Finally, the loop from line 20 computes the flow-level delay estimates produced by the model. Here, function R_{f_d} (line 23) represents a readout function that is individually applied to the hidden states of flows as they pass through a specific link ($\boldsymbol{h}_{f,l}$). The output of this function is the effective queue occupancy seen by the flow at that link. Note that this effective queue occupancy can be different for different flows depending on their traffic properties (e.g., traffic volume, burstiness). Lastly, the estimated effective queue occupancies are used to compute the final flow-level delay estimates $\hat{y}_{f_d}$. This calculation is described in more detail later (Sec. II-B2).

B. Scaling to larger networks: scale-independent features

Data-driven models typically need to see edge cases that are not commonly found in real-world production networks (e.g., link failures). This means that collecting data directly from production networks would imply testing configurations that might break the correct operation of the network. As a result, data-driven network models should be typically trained with data from controlled network testbeds. However, network testbeds are usually much smaller than real networks. In this context, it is essential for our model to effectively scale to larger networks than those seen during the training phase.

Algorithm 1 Internal architecture of RouteNet-F.

```
Input: $\mathcal{F}$, $\mathcal{Q}$, $\mathcal{L}$, $\boldsymbol{x}_f$, $\boldsymbol{x}_q$, $\boldsymbol{x}_l$
Output: $\hat{y}_{f_d}$
 1: for each $f \in \mathcal{F}$ do $\boldsymbol{h}_f^0 \leftarrow HS_f(\boldsymbol{x}_f)$
 2: for each $q \in \mathcal{Q}$ do $\boldsymbol{h}_q^0 \leftarrow HS_q(\boldsymbol{x}_q)$
 3: for each $l \in \mathcal{L}$ do $\boldsymbol{h}_l^0 \leftarrow HS_l(\boldsymbol{x}_l)$

 4: for t = 0 to T-1 do                                        ▷ Message Passing Phase
 5:     for each $f \in \mathcal{F}$ do                        ▷ Message Passing on Flows
 6:         $\Theta([\cdot,\cdot]) \leftarrow FRNN(\boldsymbol{h}_f^t, [\cdot,\cdot])$       ▷ FRNN Initialization
 7:         for each $(q,l) \in f$ do
 8:             $\boldsymbol{h}_{f,l}^t \leftarrow \Theta([\boldsymbol{h}_q^t, \boldsymbol{h}_l^t])$      ▷ Flow: Aggr. and Update
 9:             $\widetilde{m}_{f,q}^{t+1} \leftarrow \boldsymbol{h}_{f,l}^t$          ▷ Flow: Message Generation
10:         $\boldsymbol{h}_f^{t+1} \leftarrow \boldsymbol{h}_{f,l}^t$
11:     for each $q \in \mathcal{Q}$ do                        ▷ Message Passing on Queues
12:         $M_q^{t+1} \leftarrow \sum_{f \in Q_f(q)} \widetilde{m}_{f,q}^{t+1}$        ▷ Queue: Aggregation
13:         $\boldsymbol{h}_q^{t+1} \leftarrow U_q(\boldsymbol{h}_q^t, M_q^{t+1})$            ▷ Queue: Update
14:         $\widetilde{m}_q^{t+1} \leftarrow \boldsymbol{h}_q^{t+1}$            ▷ Queue: Message Generation
15:     for each $l \in \mathcal{L}$ do                        ▷ Message Passing on Links
16:         $\Psi(\cdot) \leftarrow LRNN(\boldsymbol{h}_l^t, \cdot)$               ▷ LRNN Initialization
17:         for each $q \in L_q(l)$ do
18:             $h_l^t \leftarrow \Psi(\widetilde{m}_q^{t+1})$                ▷ Link: Aggr. and Update
19:         $\boldsymbol{h}_l^{t+1} \leftarrow h_l^t$
20: for each $f \in F$ do                                       ▷ Flow: Readout
21:     $\hat{y}_{f_d} = 0$                                       ▷ Initializing the flow delay
22:     for each $(q,l) \in f$ do
23:         $\hat{d}_q = R_{f_d}(\boldsymbol{h}_{f,l}^T)/\boldsymbol{x}_{l_c}$             ▷ Queueing delay
24:         $\hat{d}_t = \boldsymbol{x}_{f_{ps}}/\boldsymbol{x}_{l_c}$                  ▷ Transmission delay
25:         $\hat{d}_{link} = \hat{d}_q + \hat{d}_t$
26:         $\hat{y}_{f_d} = \hat{y}_{f_d} + \hat{d}_{link}$     ▷ Sum of link delays along the flow
```

It is well-known that GNN models have an unprecedented capability to generalize over graph-structured data [1], [2]. In the context of scaling to larger graphs, it is also known that GNNs keep good generalization capabilities as long as the spectral properties of graphs are similar to those seen during training [3]. In our particular case, the internal message passing architecture of RouteNet-F accurately generalizes to graphs with similar structures (e.g., a similar number of queues at output ports, or a similar number of flows aggregated in queues). In practice, creating a representative dataset for RouteNet-F in small network topologies does not imply any practical limitation to then achieve good generalization properties to larger topologies.

However, scaling to larger networks often entails more aspects beyond the topology size. In particular, there are two main properties that we can observe as networks become larger: (i) *higher link capacities*, as core links of the network typically aggregate more traffic, and (ii) *different flow-level delay distributions*, as end-to-end paths are larger and they can traverse links with higher capacities. This requires devising mechanisms to effectively scale on these two features.

1) Higher link capacities: In RouteNet-F (Algorithm 1), the direct way to represent the link capacity $\boldsymbol{x}_{l_c}$ would be as an initial feature of the links' hidden states $\boldsymbol{x}_l$. However, the fact that $\boldsymbol{x}_{l_c}$ would be encoded as a numerical input value would then introduce inherent limitations to scale to larger capacity values. Indeed, scaling to out-of-distribution numerical values is recognized as a generalized limiting factor among all neural network models [4], [5].

Our approach is to exploit particularities from the networking domain to find scale-independent representations that can define link capacities and how they relate to other link-level features that can affect network performance (e.g., the aggregated traffic on links). Inspired by traditional Queuing Theory (QT) models, we aim to encode in RouteNet-F the relative ratio between the arrival rates on links (based on the traffic aggregated in the link), and the service times (based on the link capacity). This enables us to infer the output performance metrics of the model from scale-independent values. As a result, instead of directly using the numerical link capacity values, we introduce the *link load* $x_{l_{load}}$ in the initial feature vector of links $\boldsymbol{x}_l$. Particularly, we compute the link load as follows:

$$x_{l_{load}} = \frac{1}{x_{l_c}} \sum_{f \in L_f(l_j)} \lambda_f \tag{4}$$

Where λ_f is the average traffic volume of the flows that traverse the link l_j, and x_{l_c} is the link capacity. In other words, we compute the link load as the summation of all the traffic that would traverse the link without considering possible losses, and then divide it by the link capacity. Then, through the iterative message passing process, the GNN model should be able to update the load values after estimating the losses.

2) Different flow-level delay distributions: The previous mechanism enables to keep scale-independent features along with the message-passing phase of our model (loop lines 4-19 in Alg. 1), while it is still needed to extend the scale independence to the output layer of the model. In this report, we focus on scale-independence for the case of predicting flow mean delays. Note that in larger networks, delay values can vary with respect to those seen during training in smaller networks. This is because flows can cross links with higher capacities, or because flows can potentially traverse larger paths. This again poses the challenge of generalizing to out-of-distribution delay values. RouteNet-F overcomes this potential limitation by inferring delays indirectly from the mean queue occupancy on forwarding devices. Specifically, the model infers the flow delay as a linear combination of the estimated queuing delays (line 23) and the transmission delays after crossing a link (line 24).

We call the values produced by the R_{f_d} function the *effective queue occupancy*, which is defined as the mean queue occupancy experienced by a given flow f_i as it passes through a specific forwarding device. More precisely, this value is the average number of bits that have to be served on a specific output port before the packets of flow f_i are transmitted. As an example, let us consider the case of packets from a flow with low priority, which are mapped to low-priority queues. If forwarding devices implement a Strict Priority scheduling policy with several queues, the effective queue occupancy seen by those low-priority packets should include all the bits to be served in the queues with higher priority.

The prediction of this effective queue occupancy — instead of directly predicting delays — helps overcome the practical limitation of producing out-of-distribution delay values with the readout function R_{f_d}. In this case, the values produced by R_{f_d} are bounded between 0 and the maximum buffer size at output ports of forwarding devices, which is independent of the network size.

Lastly, RouteNet-F produces flow-level delay predictions $\hat{y}_{f_d}$ by combining the estimated queueing and transmission delays. The queueing delay $\hat{d}_q$ is indirectly estimated by using the effective queue occupancies (in bits) on queues for a particular flow. Particularly, queue occupancy values are estimated by the readout function $R_{f_d}(\boldsymbol{h}_{f,l}^T)$. Then, they are divided by the capacity of the link connected to the output port x_{l_c} to eventually produce a queuing delay estimate $\hat{d}_q$. Likewise, the transmission delay $\hat{d}_t$ is computed by dividing the mean flow packet size $x_{f_{ps}}$ by the link capacity x_{l_c}. With this, RouteNet-F estimates the delay of a flow after passing through a specific forwarding device and a link ($\hat{d}_{link}$):

$$\hat{d}_q = \frac{R_{f_d}(\boldsymbol{h}_{f,l}^T)}{x_{l_c}} \tag{5}$$

$$\hat{d}_t = \frac{x_{f_{ps}}}{x_{l_c}} \tag{6}$$

$$\hat{d}_{link} = \hat{d}_q + \hat{d}_t \tag{7}$$

Hence, we can compute end-to-end flow delays as the sum of all the link delays $\hat{d}_{link}$ along the flows (loop lines 20-26 in Algorithm 1).

III. RouteNet-F implementation for the Graph Neural Networking challenge

We provide an implementation of RouteNet-F in TensorFlow [6]. Note that participants of the Graph Neural Networking challenge must keep the predefined values of this implementation for training their own models.

In this implementation, we set the size of all hidden state vectors ($\boldsymbol{h}_f$, $\boldsymbol{h}_q$, $\boldsymbol{h}_l$) to 32 elements, and the number of message passing iterations to T=8. We implement $FRNN$, $LRNN$, and U_q as Gated Recurrent Units (GRU). Functions HS_f, HS_q and HS_l are implemented as 2-layer fully-connected neural networks with 32 units and ReLU activation functions in each layer. Similarly, R_{f_d} is implemented as a 3-layer fully-connected neural network, each layer with 16 units. Hidden layers implement a ReLU activation function, an the output layer has a linear activation function.

The training is fixed to 20 epochs, with 2,000 steps per epoch. Note that the training dataset must have a maximum of 100 samples. Also, we set the Mean Absolute Percentage Error (MAPE) as loss function, and use an Adam optimizer with an initial learning rate of 0.001.

References

[1] P. W. Battaglia *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.
[2] J. Zhou *et al.*, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[3] L. Ruiz, L. Chamon, and A. Ribeiro, "Graph neural networks and the transferability of graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[4] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, "Exploring the landscape of spatial robustness," in *International Conference on Machine Learning*, 2019, pp. 1802–1811.

[5] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019.

[6] C. Güemes-Palau, M. Ferriol-Galmés *et al.*, "Improving network digital twins through data-centric ai," https://github.com/BNN-UPC/GNNetworkingChallenge/tree/2022_DataCentricAI, 2022.