INTERNATIONAL TELECOMMUNICATION UNION

TELECOMMUNICATION STANDARDIZATION SECTOR

STUDY PERIOD 2017-2020

GNN CHALLENGE

Original: English

Question(s):	N/A	December 8, 2023

INPUT DOCUMENT

Source:	Team Yanbiyu	
Title:	Graph Neural Networking Challenge 2023 - Creating a Network Digital Twin with Real Network Data	

Contact:	Blessed Guda, Carnegie Mellon University, Africa	Email: **blessedg@andrew.cmu.edu**

Contact:	Carlee Joe-Wong Carnegie Mellon University (Team Mentor)	Email: **cjoewong@andrew.cmu.edu**

1.0 PROBLEM STATEMENT

Advances in network technologies are driving new network applications such as Autonomous Vehicles, Augmented and Virtual Realities. These new applications will lead to massive amounts of connected devices, which increases the complexity and cost of managing communication networks. Network modelling has been used to reduce costs by using digital representations of the network to aid in network management and planning. Conventionally, network modelling is achieved using network simulators (like Omnet++ and NS2) [1], emulation or using Queuing Theory. Network simulators that work at the packet level demonstrate a high level of accuracy but are not scalable due to the high simulation time. This is because the simulation time scales linearly with the number of packets. Meanwhile, network emulators prioritize speed but often compromise accuracy, rendering them impractical. A Network Digital Twin (NDT) goes beyond a simple network model as it uses data from the network to mirror the network's behaviour and performance.

Recent advances in networking have led to the development of powerful Graph Neural Networks (GNNs) that can mimic complex network environments effectively. These innovations have enabled the creation of lightweight and real-time NDTs. However, the absence of real-world data has hindered the understanding of how these models perform in authentic network scenarios. The 2023 edition of the Graph Neural Networking challenge aims to address this gap by tasking participants with building a GNN-based Network Digital Twin using real network data. The challenge involves modelling traffic, topology, and configuration to estimate network performance accurately, specifically focusing on delay. Participants were required to predict the mean delay for each flow based on network topology, flow packet traces, and routing configuration.

2.0 Dataset Description

The testbed network is made up of 2 switches and 8 Huawei routers (Net Engine 8000 M1A), a traffic generator[1] and a traffic capture device. These devices and the interconnecting links are shown in Figure 1. Each of the routers has six ports and each switch has 48 ports. Each router is connected to a neighbouring router through a switch, with a maximum node degree of 5 because one of the six ports is reserved for traffic generation.

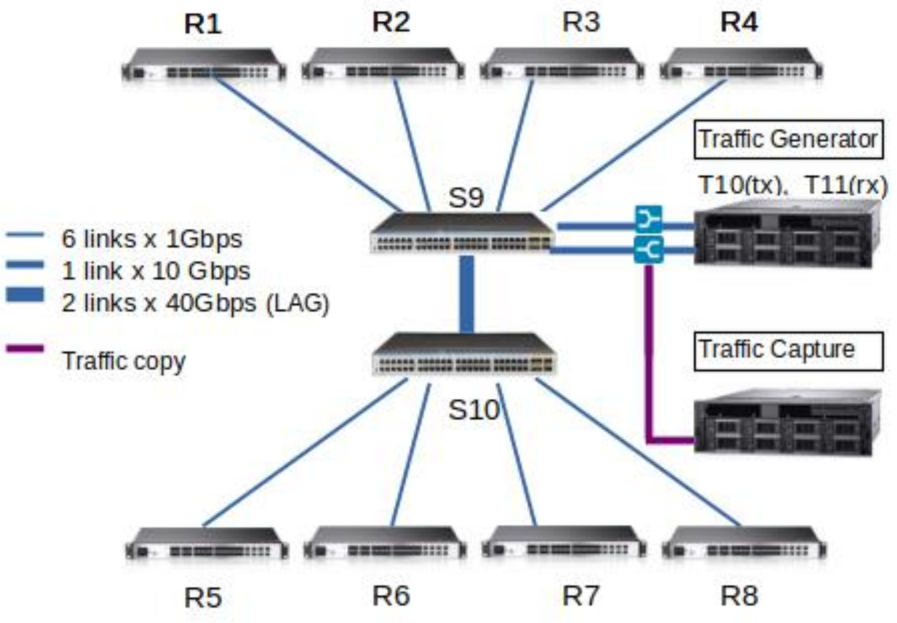

Figure 1.0: Network devices and links in the testbed. [2]

The dataset was generated from 11 random topologies of the devices across 10 different routing configurations. Each topology is made up of 5 to 8 nodes. During each experiment, traffic is generated at the traffic generator and sent to the source node. The traffic is then routed from the source to the destination node (which is always the traffic generator). The traffic capture device captures the packet traces of the traffic which is used to record 18 performance metrics relating to the delay, jitter and the number of packets dropped. This research project only focuses on three metrics, which are the mean-per-flow delay, the jitter and the number of packets dropped. This is illustrated in Figure 2 below with traffic flowing from the Traffic Generator (TG) to R1-S9-S10-R5-S10-S9-TG. Each experiment is 10 seconds long, however, only the last 5 seconds were recorded because the network is in a transient state in the first five seconds.

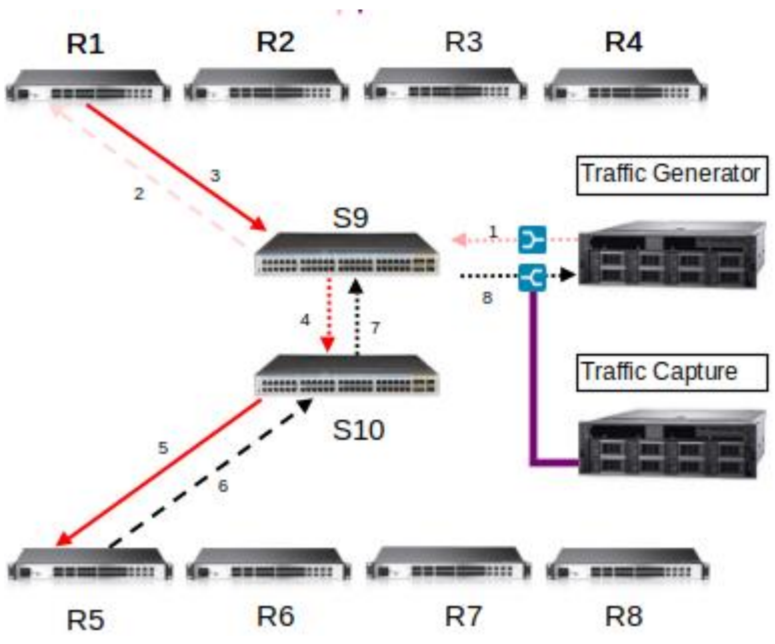

Figure 2.0: Flow of traffic during each experiment

During traffic generation, the packets are either sent in Constant Bit Rate (CBR) or Multi-burst mode (MB). In the constant bit rate, traffic is sent in short constant time intervals while in multi-burst mode packets are sent in bursts defined by an inter-burst gap time and the number of packets per burst. However, during each flow, the packet size is constant in both modes. For each configuration of the network (a specific topology and routing configuration), several sets of flows of different rates are sent and the corresponding performance metrics are recorded.. The total size of the dataset generated is 400GB across 4388 unique network configurations.

3.0 Data Preprocessing and Feature Extraction

From each flow in the packet traces, the data in Table 2.0 are extracted.

Table 2.0 Data extracted from the packet traces.

Parameter	**Description**
Flow traffic	Average bandwidth used by the flow
Flow packets	Total packet size in the flow
Link capacities	Capacities of the links (bps)
Flow packet size	The constant packet size of a flow (bits)
Flow type	CBR or MB (0 or 1)
Links	The indices of the links as traversed by the flow

Mean interpacket gap	The mean time interval between successive packets in the flow
Variance of interpacket gap	The variance of the time interval between successive packets in the flow
Packets dropped	The ratio of the number of packets dropped to the total packets sent
Mean per flow delay	The average delay of all packets in the flow
Jitter	The variance of the delay of packets in the flow

During this feature extraction process, the maximum and minimum values of the flow traffic, packets, and packet sizes are saved for min-max normalization during training and inference. This normalization helps ensure scalability to larger unseen networks and also ensures that feature values do not outweigh each other.

4.0 Model Architecture

RouteNet-Fermi[3] which is the state-of-the-art GNN model for building Network Digital Twins was used. RouteNet-Fermi was proposed as an improvement on TwinNet [4]. Both models were proposed based on the solutions from the previous editions of the GNN challenge organized by the Barcelona Neural Networking Center [5]. Both the TwinNet and RouteNet model the network by considering three main components the links, queues and the paths traversed by the flow. The goal is to model the cyclic dependency that exists between the three components. This is because a path is defined by links and queues, and the state of a queue is dependent on the paths through it and a link is dependent on the paths through them and the queues that inject traffic into it. This is illustrated in Figure 3.0 and defined by equations 1-3.

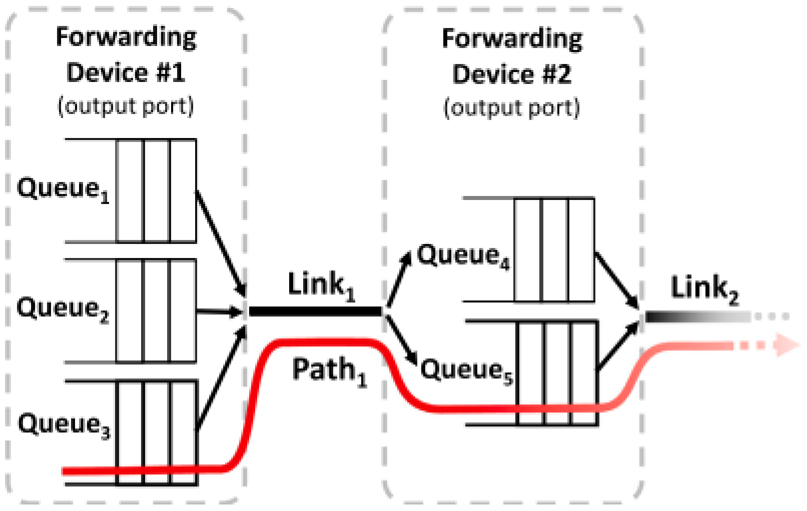
Forwarding Device #1 (output port)
Forwarding Device #2 (output port)
Queue1
Queue2
Queue3
Queue4
Queue5
Link1
Link2
Path1

Figure 3.0: The cyclic dependency between queues, paths, and links[4].

$$state_{fi} = F_f(state_{q1}, state_{l1} \dots \dots ..) \qquad (1)$$

$$state_{qi} = F_q(state_{f1}, state_{f2} \dots \dots ..)(2)$$

$$state_{li} = F_l(state_{q1}, state_{q2} \dots \dots ..)(3)$$

Where f, q *and* l are the flows, queues and links and $state_{fi}$ is the state of flow i.

Like the TwinNet, the training procedure of the RouteNet learns from the network data in three stages, namely; initialization, message passing and read-out phases.

i. **Initialization Phase**: Unlike the TwinNet that initializes the hidden states $state_{fi}$, $state_{qi}$ and $state_{li}$directly with the initial values from the network data (queue capacity, link capacity and traffic volume), RouteNet uses a learnable embedding of these values. This embedding is learnt with a two-layer Feed Forward Neural Network, whose output dimension is a variable parameter. For the flows, the initial state is initialized as the embeddings of the flow traffic, packets, packet size and flow type. For the links, the initial state is initialized with the embeddings of the link capacities traversed and the ratio of the flow traffic and link capacities (this additional term is to help in improving scalability to larger networks). The dataset provided no information about the queues is provided. Therefore, the state of the queues is not computed or used.

ii. **Message Passing Phase:** During each iteration of the messaging step, the state of a path is updated using a Gated Recurrent Unit (GRU) that uses the states of the links it passes through. This is then aggregated to get a message from the paths. The state of each link is updated using the message from each flow that injects traffic into that link. The number of iterations of the messaging step is a hyper-parameter to be tuned.

iii. **Readout Phase:** The readout phase is used to make predictions of the network performance metrics using the hidden states of the network path. The readout is a 2-layer feedforward neural network. To improve the scalability, instead of predicting the mean-per-flow delay directly (like TwinNet), it is accumulated from each node instead of reading out from the final hidden state of the destination node. Also, the delay is not predicted directly, instead, it is derived from predicting a virtual effective occupancy of the queue of a forwarding device. The effective occupancy is then divided by the capacity of the link at the output of the queue to estimate the delay. Using this approach makes the predictions to be scalable to larger networks that have a longer end-to-end path. The packets dropped are predicted using the final hidden state of the flow with a sigmoid activation because it is a fraction of the total packets sent. Similarly, the jitter is predicted with a readout network with a linear activation.

We used the GELU (Gaussian Error Linear Unit)[6] instead of the ReLU (Rectified Linear Unit) activations used in the hidden layers of the RouteNet architecture. The GELU activation function is described in Equation 4,

$$GELU(x) = x\Phi(x) \qquad (4)$$

Where $\Phi(x)$ is the $(P(X \leq x))$ where $X \sim \mathcal{N}(0,1) = \; x \cdot \frac{1}{2}\left[1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$

The GELU weights inputs by their values rather than their sign as is done by ReLU. This gives information on how large the input data is compared to other inputs. The GELU gives a smoother function compared to the ReLU which helps during backpropagation as it provides a continuous gradient. Also, the GELU is more popularly used in NLP that involves sequences of data which is like the problem being solved in this project. Additionally, the GRUs used to update the states of the links and flows were regularized using an L2 penalty to improve the generalization of the model. The parameters used are summarized in Table 3.0

Table 3.0: Summary of hyperparameters used.

Parameter	**Value**
Hidden Layer activation functions	GELU
Path State Dim	32
Hidden State Dim	32
Regularization [Link Update RNN]	L2-regularization with alpha = 0.01

The team has also provided an implementation of RouteNet fermi in Pytorch. This can be a helpful resource to the research community [7]. Also, our implementation in Pytorch has the advantage of allowing back-propagating the loss in mini-batches instead of just using batch size. However, this has the disadvantage of being slower than the TensorFlow equivalent due to the TensorFlow implementation running on lazy execution mode and the use of a RaggedTensor data type.

5.0 Model Training and Evaluation

The model was trained using a learning rate scheduler that monitors the validation MAPE with a patience interval of 5 epochs. The learning rate was initialized at 0.01 with an Adam optimizer. For the competition, the model was trained for only predicting the mean per-flow delay. The model stopped early at 24 epochs, achieving a training, validation and testing MAPE of **36.42%, 41.93%** and **43.39%.** The team further experimented predicting the three-performance metrics at once by backpropagating the average of the three losses from three readout networks. The MAPE results are summarized in Table 4.0

	Train	**Validation**
Mean-per-flow delay	61.20	62.17
Jitter	54.96	55.84
Packets dropped	35.72	37.21
Average	50.020	51.30

Other key learnings by the team through ablation experiments are summarized in the Appendix section.

References

[1] "TRex." https://trex-tgn.cisco.com/ (accessed Nov. 24, 2023).

[2] "dataset - BNN-UPC." https://bnn.upc.edu/challenge/gnnet2023/dataset/ (accessed Nov. 24, 2023).

[3] M. Ferriol-Galmés *et al.*, "RouteNet-Fermi: Network Modeling with Graph Neural Networks," *IEEE/ACM Trans. Netw.*, Dec. 2022, doi: 10.1109/TNET.2023.3269983.

[4] M. Ferriol-Galmés *et al.*, "Building a Digital Twin for network optimization using Graph Neural Networks," *Comput. Networks*, vol. 217, p. 109329, Nov. 2022, doi: 10.1016/J.COMNET.2022.109329.

[5] J. Suárez-Varela *et al.*, "The Graph Neural Networking Challenge: A Worldwide Competition for Education in AI/ML for Networks," *Comput. Commun. Rev.*, vol. 51, no. 3, pp. 9–16, Jul. 2021, doi: 10.1145/3477482.3477485.

[6] D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)," Jun. 2016, Accessed: Nov. 25, 2023. [Online]. Available: https://arxiv.org/abs/1606.08415v5

[7] https://github.com/gblessed/gnn_challenge/tree/main

APPENDIX

Other key learnings by the team for other experiments carried out.

1. Minimizing the MSE instead of MAPE did not help out.
2. Adding more layers to the GRUcell blocks, readouts or embedding layers does not help out.
3. Using an LSTMcell instead of GRUcell did not help out.
4. The team is also currently trying out experiments with transformer encoders.
   The motivation of using transformers is for the model to be able to identify the links/nodes in the path that contribute more to the cumulative delay. To this, two experiments were carried out:
   i. Formulating the problem as a sequence "classification" problem.
      For an input flow that traverses k links, we get the link embeddings, $L = \{h(L_1), h(L_2) \ldots . h(L_k)\}$ and an input flow embedding, $\{h(F_1)\}$ we concatenate the flow embeddings to the link embeddings to form a sequence. The sequence was passed through an encoder to get the hidden state that is passed to the readout to make predictions. The model started at a validation MAPE of 56.8 after epoch 1, but takes a very long time to train. We were only able to train for 5 epochs and the loss was decreasing but quite very slowly.
   ii. To replace only the path state GRUcell in the Routenet architecture with the transformer decoder, while re-using the link GRUcell. << In progress>>

5. It only took the model 24 epochs to get to the reported performance level [validation MAPE = 41.23%]
6. The team has also provided an implementation of RouteNet fermi in Pytorch. This can be a helpful resource to the research community [2]. Also, our implementation in Pytorch has the advantage of allowing back-propagating the loss in mini-batches instead of just using batch size =1. However, we couldn't find an equivalent TensorflowRaggedTensor data type in Pytorch made us to use several for loops to achieve the same functions implemented in torch.