Beyond Mean: Spatio-Temporal Modeling of Queue Utilizations and Flow Latencies Using T-GNNs

Max Helm, Benedikt Jaeger, Christopher Pfefferle, Georg Carle
Department of Computer Engineering, *Technical University of Munich*
{helm, jaeger, pfefferl, carle}@net.in.tum.de

***Abstract*—Network planning and control require precise, reliable, and dynamic digital network models to easily obtain performance metrics. One central performance metric in any network is the end-to-end latency of connections which can be inferred from queue utilizations along its path. Models take a variety of forms: simulation, emulation, stochastic and deterministic formal methods, and machine-learning-based or -assisted approaches. Simulation and emulation require either too much computational time or too many hardware resources, while formal methods often have a high computational complexity leading to poor scalability. Machine-learning-based methods scale better to larger problem spaces, however, current approaches mainly concentrate on mean performance metric predictions. We show that such an approach can be extended to predict queue utilization and end-to-end latency behavior over time in dynamic networks. This is achieved by utilizing Temporal Graph Neural Networks (T-GNNs) which can model spatio-temporal dependencies. The approach achieves a mean queue utilization error of 5.5% and a flow-level end-to-end latency MARE of 5%-55% depending on time resolution over 100 random topologies. We show that this approach outperforms a non-temporal, static Graph Neural Network (GNN) on the same task in terms of capturing dynamic network behavior such as queue build-up and draining. The approach performs similar to related work while increasing flow rates by up to three orders of magnitude—this improvement is bought with a trade-off in supported scheduling mechanisms and traffic patterns. Our results show that such a T-GNN approach can be useful for performance modeling of high data rate flows in dynamic networks.**

***Index Terms*—graph neural networks, time series, temporal, latency prediction**

I. Introduction

Modeling network behavior plays a central role in network planning, maintenance, and performance evaluation. It is of specific importance to have models that can accurately and precisely react to changes in the network. Furthermore models should be able to compute performance metrics in adequate time, such that the network can be dynamically re-configured on performance metric- or network component changes. This is a typical pattern used for digital twins: The model (as the instance of a digital twin) is fed with dynamic information from the network, computes performance metrics, and returns this information to the network, which can perform adjustments based on this data.

A performance metric of particular interest is end-to-end latency of flows as well as congestion at all device queues in the network. The metrics can be obtained at two levels: either as a scalar with assurance on the validity, or as a timeseries of metrics.

At both levels, these can be obtained using different approaches. The scalar can be obtained, for example, using Queuing Theory (QT) [1] which provides a mean value, deterministic network calculus [2], [3] which provides a worst-case upper bound, stochastic network calculus [4] which provides a worst-case upper bound with a violation probability, extreme value theory [5] which provides expected worst cases or arbitraty percentiles, and machine-learning-based approaches [6] which can, e.g., provide mean values with a certain accuracy.

The timeseries can be obtained using, for example, simulation with a network simulator such as *ns-3*[1] or *OmNET++*[2], network emulation with *Mininet*[3] or *Containernet* [7], or live measurements on the system of interest.

We develop a machine-learning-based model that predicts timeseries of congestion and latencies in dynamically changing networks. Our solution relies on GNNs with an extension into the temporal domain to model time-dependent correlations. This approach is called T-GNNs. Compared to existing works, we work on a more diverse set of random topologies, we use flows with data rates three orders of manitude larger. To be computationally feasible, we make trade-offs in the number of traffic patterns and scheduling algorithms.

Our contributions are as follows:

1) We implement an approach to extract queue utilizations with ms granularity in simulated networks with congestion which increases flow latencies. This is applied to 300 randomly generated networks with 10,000 timesteps each.
2) We implement, based on related work, a spatio-temporal T-GNN model that can predict queue utilizations over time.
3) We identify the most influential factors for the decision making of the T-GNN model
4) We compare a static GNN approach on the same task.
5) We provide open-source access to our temporal dataset and the T-GNN implementation[4].

Section II provides background knowledge and shows related work. Section III details our approach and Section IV evaluates the performance of the approach and compares it to a static baseline. Section VI explains how to reproduce our results, before Section VII concludes with future work.

[1] https://www.nsnam.org/ [2] https://omnetpp.org/ [3] http://mininet.org/
[4] https://github.com/tgnn-test/dataset

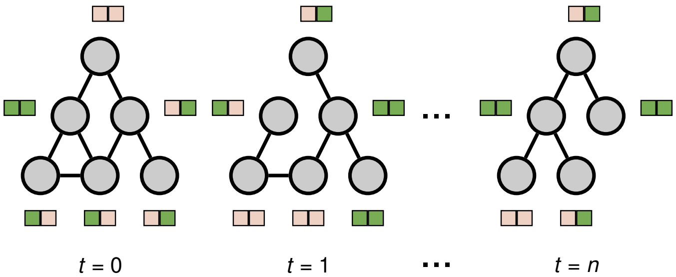


Fig. 1: Graph with time-variant features and topology at different time steps t. Edges and nodes can be added or removed, and node features (red and green 2×1 vectors) can change their values. Adapted from [12]

II. Background and Related Work

The following covers relevant concepts of GNNs, T-GNNs, and latency modeling. Furthermore, it provides an overview of related work in these three areas.

A. Background

GNNs [8] are a neural network approach that can work directly on graph-structured data, utilizing the permutation invariance property of graphs. This means that different representations of the same graph lead to the same results, which is not fundamentally true for other approaches, such as Convolutional Neural Networks for image processing. Geometric deep learning approaches, such as GNNs, can be considered generalizations of many of these other neural network approaches [9]. The input graph is defined as $G = (V, E)$ where V is a set of vertices and E is a set of edges. Each vertex and edge is associated with a vector of features. These inputs are encoded into two matrices, the feature matrix, and the adjacency matrix. During the training of GNNs, a message passing step and an aggregation step are performed. The message passing exchanges information along the edges of the graph, the aggregation aggregates the exchanged information into a hidden state at each vertex. The aggregation is typically an invariant function, e.g., mean, sum, or maximum. However, more sophisticated methods exist, such as attention-based or Long short-term memory (LSTM)-based approaches [10].

T-GNNs are an extension from the purely spatial domain to a spatio-temporal domain [11], [12]. They allow predictions on time-variant graphs with time-variant features. Figure 1 shows an example of a time-variant graph with time-variant features. Nodes and edges can be removed or added, and feature vectors can be changed over time. The input graph is extended compared to GNNs to a set of graphs including time steps: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ where $\mathcal{T}$ is a set of time steps, indicating a period of time. T-GNNs typically utilize LSTM cells to capture long- and short-range temporal dependencies. An LSTM cell outputs a hidden state and a cell state, which are used as inputs in the next iteration. Together with the forget gate, they allow capturing temporal dependencies.

Latency modeling in networks can be approached using a myriad of approaches. Latencies over time can be obtained using three approaches. First, network simulators, such as *ns-3* or discrete event simulators with extensions for communication networks, such as *OmNET++*. Second, emulation such as *Mininet* or *Containernet*. Third, measurements on live systems.

Mean latencies can be obtained using QT which requires the sending behavior of devices to conform to specific traffic models. Mean latencies can also be obtained using machine-learning-based approaches such as *RouteNet* [6], [13], [14]. Worst-case upper bounds on latencies can be obtained by employing formal methods, such as network calculus.

B. Related Work

Spatio-temporal predictions have been applied to traffic forecasting on road networks using different T-GNN architectures [15]–[20]. These approaches mostly rely on fixed road networks, whereas we generate 300 random network configurations.

Yao et al. propose a GNN-based architecture for the prediction of cumulative flow throughput and evaluate it on real-world data of a backbone network [21]. *Wang et al.* propose a timeseries-based GNN architecture to predict the temporal behavior of Call Detail Records in 5G networks and evaluate their approach on real-world data [22]. Our approach predicts performance metrics on a more granular level while relying on simulated data of 300 randomly generated topologies.

Taheri and Berger-Wolf propose the DyGrAE T-GNN architecture [23]. Our approach uses a slightly modified version thereof, as explained in Section III-A.

RouteNet is a GNN-based model predicting end-to-end latencies of network paths under a variety of traffic and scheduler models [6]. They achieve small MAPE values while maintaining scalability to networks of up to 300 nodes. Our approach uses smaller topologies and more homogeneous traffic and scheduler models but introduces temporal latency predictions instead of mean predictions. Furthermore, our approach supports flows with sending rates two to three orders of magnitude larger.

Wang et al. propose an extension to RouteNet to predict temporal end-to-end latencies of network paths [24]. They rely on a factorization approach, i.e., they use the outputs of previous time steps as inputs to a GNN model. The architecture consists of an encoder block that creates embeddings per timestep. The embeddings are concatenated with the GNN outputs of the previous timestep. This is then fed into the GNN block. The output of the GNN block is used for concatenation to the next timestep and as an input to a final decoder block that acts as a latency readout function. Our approach utilizes a dedicated temporal component with an LSTM cell as shown in Figure 2. Furthermore, we don't manually use outputs of previous timesteps as inputs for future timesteps. Our approach supports flows with sending rates two to three orders of magnitude larger. Additionally, we provide access to our temporal dataset as well as the model architecture and tranining scripts.

III. Methodology

This section provides an overview of the T-GNN architecture, graph representation, dynamic graph representation, the

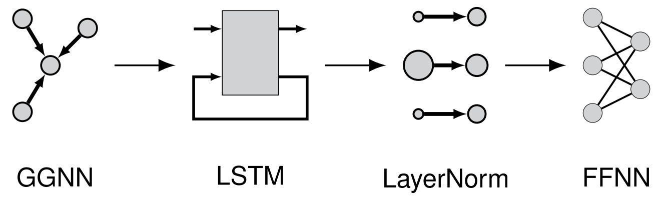


Fig. 2: Structure of the Temporal GNN

Layer	**Type**	**Size**
Spatial	GGNN	$53248_w + 384_b$
Temporal	LSTM	$98304_w + 1024_b$
After Cell	LayerNorm	$128_w + 128_b$
Output	Fully Connected	$128_w + 1_b$
		$\sum$ 153,345 parameters

TABLE I: Temporal GNN layers with weight w and bias b parameters

dataset generation, and the training process.

A. Temporal GNN Architecture

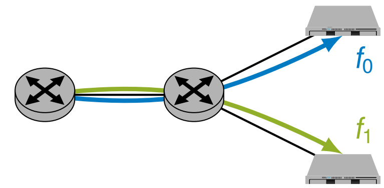
The temporal GNN architecture consists of two main components: a Gated Graph Neural Network (GGNN), and an LSTM. The GGNN is responsible for modeling the spatial aspects while the LSTM is responsible for modeling the temporal aspects, resulting in an architecture capable of modeling spatio-temporal behavior. This architecture is supplemented with a normalization layer to deal with vanishing gradients, and a fully connected feed-forward network to decode the results. A representation of the data flow through these components is shown in Figure 2. The data flow is implemented by the `forward` function. This function additionally includes a LeakyReLU activation function and a dropout layer between the normalization layer and the feed-forward network. The LeakyReLU is used over a ReLU to avoid the "dying ReLU"-problem, effectively not discriminating between inputs. The dropout is used to avoid overfitting and increase generalization capabilities by randomly de-activating connections. Furthermore it includes a Hardtanh activation function with a range of zero to one after the final feed-forward network. This is used to scale the output to a normalized queue utilization value. Table I shows a detailed view of the distribution of model parameters between all layers.

The model uses the *DyGrEncoder*[5] recurrent graph convolutional layer implemented in *PyTorch Geometric Temporal*.

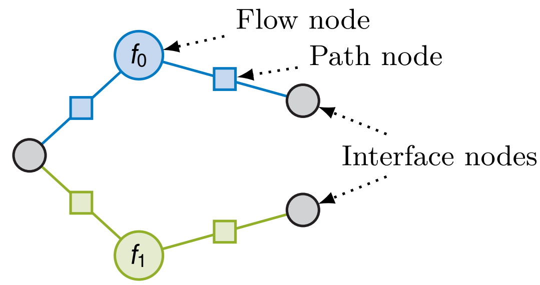
B. Graph Representation

The graph representation is a logical representation of the physical network topology and flow configurations [25], [26]. This representation is the direct input to the T-GNN. Figure 3b shows an example of such a representation for a small topology. Egress interface nodes are modeled as individual nodes, while ingress interfaces are not included. This is because delay induced by queuing typically occurs at

[5] https://pytorch-geometric-temporal.readthedocs.io/en/latest/modules/root.html#torch_geometric_temporal.nn.recurrent.dygrae.DyGrEncoder



(a) Topology with flows



(b) Graph representation with flow-, path order-, and egress interface nodes

Fig. 3: Topology and corresponding graph representation

the egress interfaces [27]. Each flow is modeled as a separate node and is connected to each egress interface it traverses. Between flow and interface nodes, we included path nodes which indicate the direction of transmission of the flow. The direction is encoded as a scalar feature of the path nodes. This graph representation supports the message passing step by grouping functionally dependent nodes topographically close together. Interface nodes connect flows that interfere with each other at this interface over four hops. This means a message passing step with four unroll operations exchanges information between two interfering flows.

C. Temporal Representation

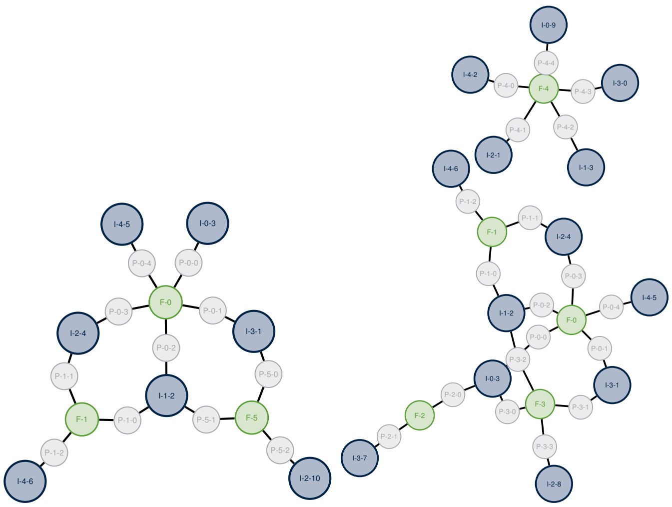
Temporal behavior, such as variations in sending rate of flows, is encoded by adding and removing edges in the graph representation at the respective time steps. This leads to the generation of a dynamic graph.

A dynamic graph is a mathematical structure $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ where $\mathcal{V} = \{V(t)\}_{t \in \mathcal{T}}$ is a collection of node sets, $\mathcal{E} = \{E(t)\}_{t \in \mathcal{T}}$ is a collection of edge sets, and $\mathcal{T}$ is a time span. For each $t \in \mathcal{T}$ a graph snapshot is defined as $G(t) = (V(t), E(t))$, equivalent to the static definition. This static graph represents one time step of the dynamic graph. [28]

Figure 4 shows two snapshots of such a dynamic graph using our graph representation to encode logical dependencies. We can observe that the size of the graph changes over time. Furthermore, the graph is fully connected at the first timestep, while being disconnected at the second one. A subgraph of a disconnected graph thereby encodes a realm in which flows interact by traversing the same egress interfaces. Flow nodes in disconnected parts of a graph do not interact at any point in the network.

D. Datasets

We generated data utilizing the ns-3 network simulator [29]. The dataset consists of randomly generated networks with



(a) Graph snapshot at $t = 3.54$ (b) Graph snapshot at $t = 6.45$

Fig. 4: Two snapshots of a dynamic graph using our graph representation. Indicates the logical dependencies of a single topology at different time steps. Blue nodes show egress interfaces, green nodes flows, and grey nodes paths.

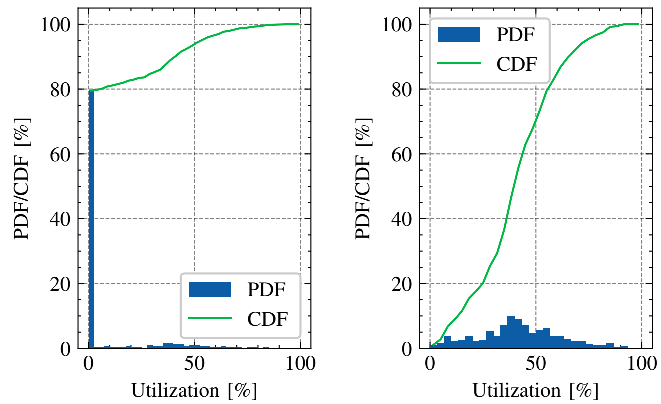
Parameter	Median	Min	Max	Unit
Number of nodes	10.0	7	12	—
Number of links	9.0	6	11	—
Number of flows	5.0	4	6	—
Flow length	4.0	3	7	Hops
Packet size	1472.0	1472	1472	B
Link rate	50.0	10	100	$\mathrm{Mbit\,s^{-1}}$
Flow rate	29.0	10	199	$\mathrm{Mbit\,s^{-1}}$
Queue size	125.0	125	125	kB
Link delay	0.549	0.101	1.0	ms
On/off interval	1.8	0.5	3.0	s
End-to-end flow delay	36.76	0.81	196.14	ms
Packet Inter-arrival-time	0.0012	0.0010	4.10	s

TABLE II: Distributions of parameter values for randomly generated networks

random flow- and node configurations. All flows are periodic with on- and off intervals and carry UDP payloads of a fixed size. The distribution of values for each parameter is shown in Table II. Specifically, the flow rates, link rates, and on-off intervals are chosen such that at times we can observe no congestion and at other times we can observe heavy congestion between multiple flows.

To extract the labels for our datasets, we need to measure and export the queue utilization levels at different points in time. We sample the amount of data in each queue in fixed intervals of 1 ms. In conjunction with the maximum queue size, we can derive the queue utilization. Depending on the exact sampling time, queues might contain single packets, despite having zero utilization and being able to transmit packets without queueing. These instances are sanitized in a postprocessing step.

Figure 5 shows the queue utilization of egress interfaces over all topologies and timesteps. We can observe that 80% of timesteps have queues with zero utilization. This does not mean that the queues are not traversed by flows, it only means that the number of packets in the queues is less than or equal to one, i.e., the sum of rates of the traversing flows is less than the transmission speed of the interface. When only considering the non-zero utilized queues, we can observe a normal distribution of utilization between 0 and 100% as shown in Figure 5b.



(a) All queues (b) Non-zero utilization queues

Fig. 5: Mean queue utilizations in the training dataset

Since we want to model end-to-end latencies, we need to derive them from the queue utilizations. We follow a similar approach as [30] by calculating the queueing delay of a packet in a queue as shown in Equation (1) where u is the measured queue utilization, s^{max} is the maximum size of the queue, and c is the link speed.

$$d_{queueing} = \frac{u \cdot s^{max}}{c} \tag{1}$$

The end-to-end latency of a flow then consists of the sum of per-queue queueing, transmission, and serialization delays of all traversed queues as well as the propagation delay of all traversed links. This is shown in Equation (2) where P is the set of egress interfaces traversed by the flow and l_i is the link attached to interface i.

$$latency_{e2e} = \sum_{i \in P} d^i_{queuing} + d^i_{trans.} + d^i_{serial.} + d^{l_i}_{propa.} \tag{2}$$

The queue utilizations are used as ground truth for the T-GNN model and the end-to-end flow latencies are calculated in a post-processing step.

We create a total of three datasets. A training dataset to train the T-GNN, a test dataset to calculate the test loss during the training process, and an evaluation dataset to analyze the performance of the fully trained model. The sizes of the datasets are shown in Table III, we can observe a 46.67%, 20%, 33.33% split, which lays within commonly used bounds [31].

E. Training Process

Hyperparameter tuning is performed manually by evaluating the test accuracy of models with different parameter combina-

Dataset	Topologies	Snapshots	Queue Events
Train	140	2.5M	12.16M
Test	60	1.1M	5.21M
Evaluation	100	1.8M	8.69M

TABLE III: Number of topologies and time snapshots per dataset, resulting in a 46.67%, 20%, 33.33% split. Queue events are the number of ground truth labels per dataset.

tions after the first training epoch on a small subset of training data. We use the mean squared error as a loss function since we perform a regression task on the queue utilization. The model is trained for 200 epochs, after which no additional performance gains were registerable. Afterwards we select the best performing one based on loss values.

IV. Evaluation

This section evaluates the prediction quality of the fully trained T-GNN model. The evaluation is performed at the queue, flow, and network levels. Furthermore, we perform a comparison to a static GNN implemented by us, and another state-of-the-art approach. Last, we analyze the importance of different input features on the prediction quality, drawing conclusions about the learned correlations.

A. Queue Level

The T-GNN model predicts the utilization of queues on the egress interfaces of devices. Figure 7 (middle) shows the utilization labels (ground truth) and predictions of one such queue over the full period of 10 s. To the left and right are depictions of the same queue, at a higher time resolution, highlighting queue build-up and drain events. We can observe that the three queue build-up and draining events are accurately reflected in the models' predictions. When considering the higher time resolution plots of queue build-ups, we can see that the linear increase in queue utilization is approximated with a non-linear function. However, the linear function of the queue draining event is predicted as a linear function with a small offset. Both build-up and draining exhibit only small deviations at the ms resolution.

Figure 8 contrasts these observations with a queue for which the T-GNN models' predictions exhibited a larger error. We can observe some artifacts in the utilization predictions, expressed through volatile and abrupt changes in utilization. Specifically, we can observe a sudden drop in utilization in the queue draining event, before the incorrect prediction is corrected and the queue is finally drained correctly.

The queue build-up event plateaus at 40% before the T-GNN model corrects itself within 200 ms. Figure 6 shows the corresponding network configuration. The queue in Figure 8 is the egress interface of the highlighted router. We can see that four flows traverse this router, with three of them traversing this egress interface. The behavior of these three flows over time is shown in Figure 9a. The summation of their on-off periods, leading to the number of currently active flows on this interface, is shown in Figure 9b. The green dashed lines correspond to the two artifacts in the draining phase

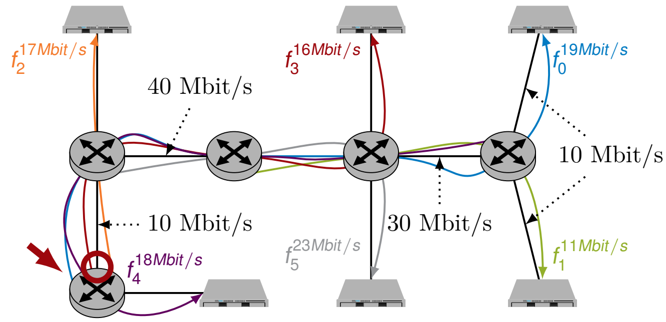


Fig. 6: Example topology with flow rates and relevant link rates. Flow rates are sending rates in On-state of the On-Off traffic model. Queue utilization of the ingress interface highlighted by the red circle is displayed in Figure 8.

shown in Figure 8. We can see that the sudden drop of queue utilization is caused by the number of active flows dropping from two to one. However, the link is still being saturated by this one flow, which the T-GNN model did not represent correctly, before correcting itself 300 ms later. The red dotted lines correspond to the artifacts during the queue build-up. We can see that the incorrect prediction of 40% utilization stems from an incorrect distribution of the utilization over these two flows. We assume that the LSTM cell "reserves" queue utilization for the second flow (f_3) that is to join f_0 shortly. The 40% utilization roughly corresponds to the ratio between the sum of the two flow rates of f_0 and f_3 and their respective rates, which is 46% to 54%.

Figure 10 is looking at a cumulative metric over each queue utilization over time of each of the 100 topologies in the evaluation dataset. The chosen metric is the Mean Absolute Error (MAE) between predicted queue utilization and the ground truth. The average MAE is 0.055, meaning on average we have an error of 5.5% in the queue utilization predictions. Furthermore, we can observe fluctuations in the MAE over time, which correlate to the cumulative transmission rates of all flows. Additionally, the peak MAE values at approximately 3.5 s, 7 s, and 8.75 s correspond to $2\times$, $4\times$, and $5\times$ the mean on-off interval of 1.75.

B. Flow Level

To evaluate the predictions of the T-GNN model on a flow level, we calculate the end-to-end latencies of flows as described in Section III. We calculate the end-to-end latency once from the predicted utilizations, and once from the ground truth utilizations. We perform this step on our full evaluation dataset of 100 topologies.

C. Network Level

At the network level, i.e., averaging the flows per topology and then averaging over all 100 topologies, we observe an MdAPE of 13.37%.

D. Comparison to Static Model

To evaluate the effectiveness of the temporal component of the T-GNN, we compare it to a static GNN. The static

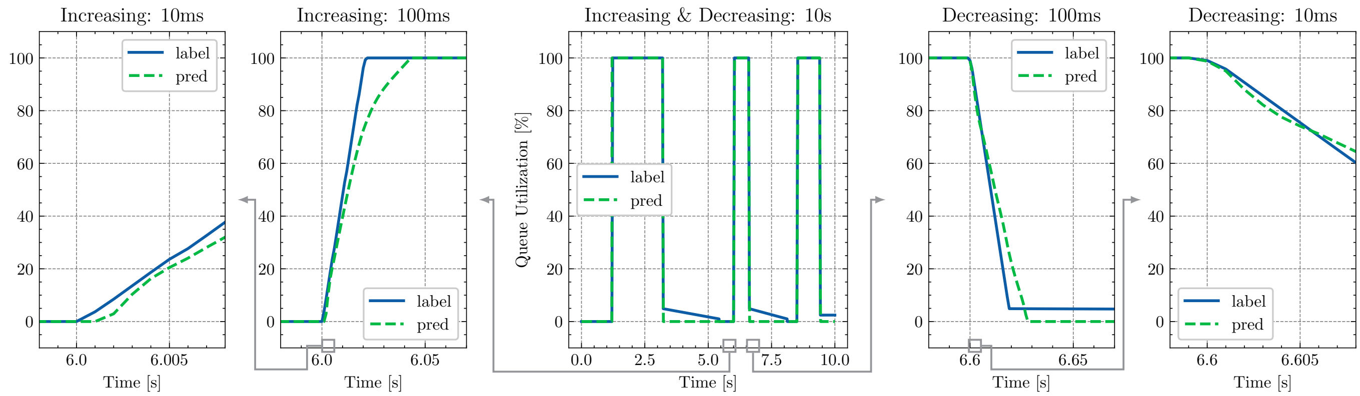


Fig. 7: Queue utilization labels and predictions over a timespan of 10 s. Highlights queue utilization increase and decrease over timespans of 100 s and 10 s.

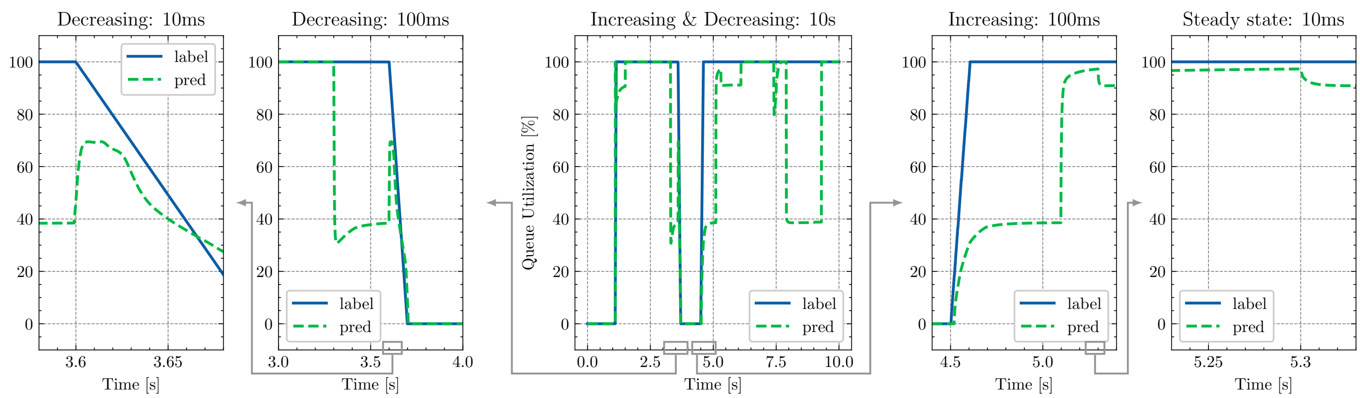


Fig. 8: Queue utilization labels and predictions over a timespan of 10 s. Highlights queue utilization increase and decrease over timespans of 100 s and 10 s.

model is trained on the same data for the same amount of epochs. Each snapshot of the dynamic graph is represented as a separate static graph.

Figure 11 shows a comparison of the performance during queue build-up and draining events. While the T-GNN model predicts the increase and decrease in utilization over time with reasonable accuracy, we can see that the static GNN model jumps from zero to full utilization. The jumps happen at the moments where the queue starts to fill and drain respectively. This shows that the temporal component is able to better capture and model these dynamics.

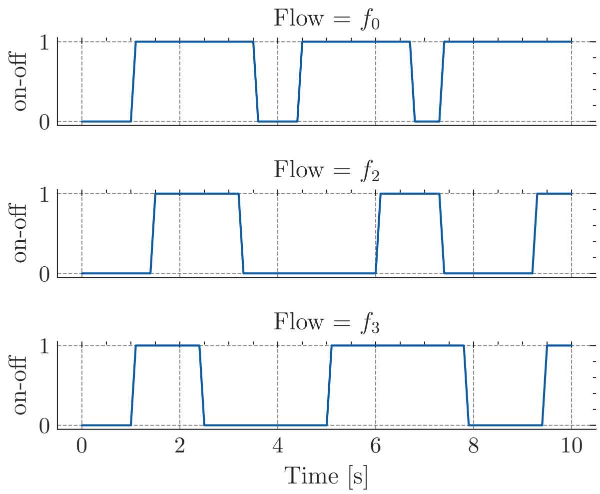
However, the overall MAE score of the static model (5.2%) is very similar to that of the T-GNN model.

An explanation for the similar performance of the two models, despite the better queue build-up and draining model of the T-GNN, are artifacts in the T-GNN models predictions that are absent in the GNN predictions. Such an artifact is shown in Figure 12. We see an increase in predicted utilization over a timespan of 20 ms which is seemingly random. Upon further investigation, such artifacts are caused by the removal of edges between flow and interface nodes (connected by path nodes). At $t = 3.09$ a flow that has not been saturating the link, and thus has not been building up a queue, stops sending. This is modeled in the T-GNN by removing the corresponding edge between flow and interface as shown in Figure 13.

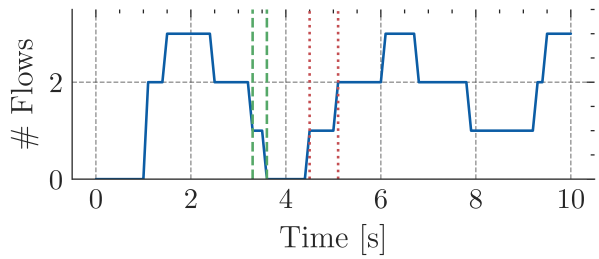
If the link were to be saturated, we would expect to see a change in queue utilization upon removal of the flow. We assume that this line of reasoning is the cause for these types of artifacts.

E. Comparison to State-of-the-Art

Comparison to state-of-the-art machine learning approaches is difficult, since to our knowledge, there is only one approach that predicts temporal latencies in computer networks [24]. However, they do not provide a reference dataset to compare against. This means that we aren't able to perform a direct comparison, instead, this is a broad placement of our approach into related work. We additionally compare to RouteNet-Erlang [14] as a close related work because it predicts mean latency values in computer networks. Table IV shows the Mean Absolute Relative Error (MARE) and MAPE scores as reported in related work compared to the MARE for our GNN and T-GNN approach. Furthermore, it includes a T-GNN model which was trained and evaluated on a different time granularity of 10 ms instead of 1 ms. The results show a decrease in error, which is to be expected since lower frequency



(a) On-off periods of the three flows



(b) Number of active flows

Fig. 9: On-off behavior and temporal overlap of the three flows traversing the egress interface of the queue utilization shown in Figure 8

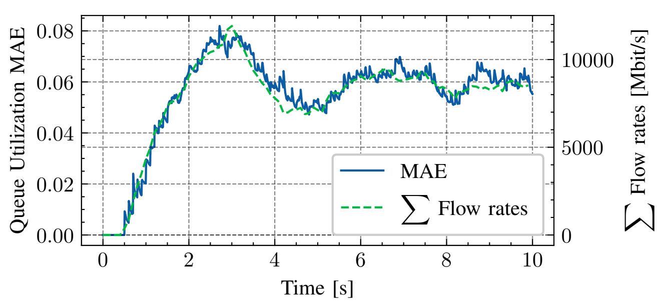
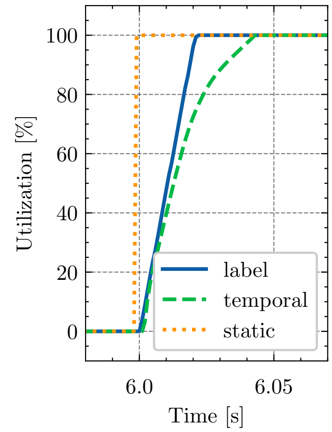


Fig. 10: The Mean Absolute Error (MAE) over time between queue utilization labels and predictions over all 100 topologies in the evaluation dataset which correlates with the sum of flow rates of currently sending flows

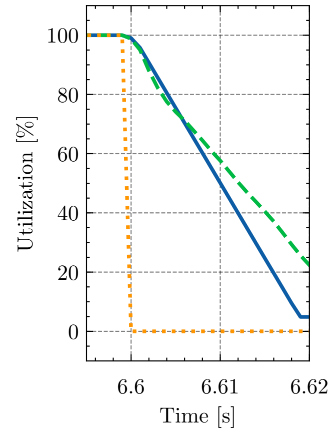
sampling of queue utilizations leads to the disappearance of certain effects. Overall, our approach can reach similar results to related work, albeit on a different dataset.

F. Comparison to Network Calculus

We compare our results to a simple network calculus model. The model estimates the worst-case backlog of each queue using the total flow analysis. The results show an infinite backlog for 80.87% of queues. The remaining 19.13% of queues exhibit queue utilizations from 2.18% to 32.78%. Since



(a) Queue build-up



(b) Queue drain

Fig. 11: Comparison of queue build-up and draining phases between temporal and static model

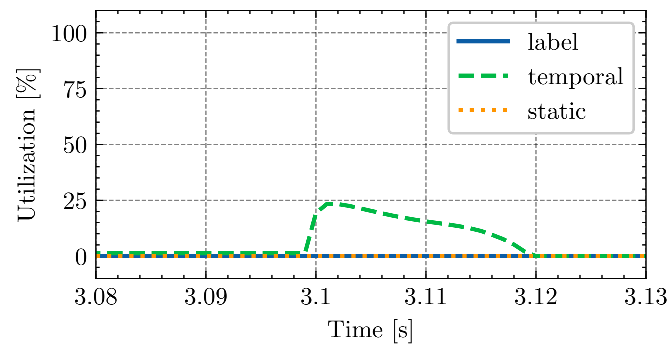
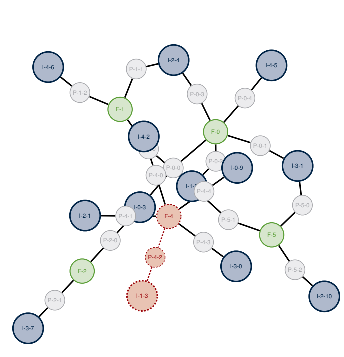
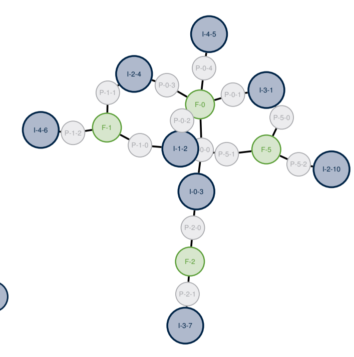


Fig. 12: Artifacts in queue utilization predictions of the T-GNN model, compared against the labels and GNN model predictions. Erroneous increase in utilization is due to a flow finishing transmission and the corresponding removal from the dynamic graph.



(a) $t = 3.09$



(b) $t = 3.11$

Fig. 13: Removal of a flow that leads to the artifact in T-GNN prediction observed in Figure 12. Responsible flow, path, and interface nodes are highlighted in dashed red.

Metric	Approach	Score	Metric
Mean			
	RouteNet-Erlang [6]	6.00% [6]	MARE
	RouteNet-Fermi [14]	6.24% [14]	MARE
Per-timestep			
	GNN (ours)	64.89%	MARE
	T-GNN$_{1\text{ ms}}$ (ours)	55.35%	MARE
	T-GNN$_{10\text{ ms}}$ (ours)	5.04%	MARE
	xNet$_{0.2\text{ ms to }5\text{ ms}}$ [24]	<5-7% [24]	MAPE

TABLE IV: Comparison of Mean Absolute Relative Error (MARE) and MAPE of different approaches. *Mean* and *Per-timestep* approaches do not compute the same metric and perform predictions on different datasets. T-GNN$_t$ means a model trained and evaluated on a time granularity of t.

the majority of backlogs are infinite, it is not possible to calculate end-to-end delays.

G. Feature Importance

The feature importance is a model-agnostic approach to reconstructing the reasoning that led to a specific machine-learning prediction. It can be derived by permuting one input feature at a time over the whole evaluation dataset, calculating the MAE of the predictions using this adjusted dataset, and calculating the difference to the baseline as shown in Equation (3). [32]

$$I_{feature} = MAE_{feature} - MAE_{baseline} \tag{3}$$

The baseline is the MAE of the non-permuted dataset. A larger difference means a bigger influence of this feature on the final prediction. Figure 14 shows the MAE and feature importance scores of the input features of our model. We can observe that the egress link bandwidth has the largest influence, while the link delay has the smallest influence. This makes sense since the egress link bandwidth is the deciding factor of whether a queue builds up or whether all incoming traffic can be served without queueing. The link delay has the smallest influence, which makes sense since the link delay does not directly correlate with queue utilization — except when flows start sending at similar times and have different link delays towards a common point of interference. In this case, the build-up phase of the queue might vary slightly. The same applies to the queue draining phase.

Node type, flow rate, and path order value have a similar, medium-sized impact. This makes sense because the flow rate has an impact on whether a queue is heavily utilized or not, however, the influence is less than that of the interface bandwidth because we have multiple flows at single interfaces. The path order value has some influence because it encodes in which direction congestion propagates along the flow path. The node type has an obvious influence by encoding which nodes are interfaces, flows, or paths. We can conclude that the T-GNN model learns meaningful dependencies between input features.

Furthermore, all features have a mostly positive impact on model accuracy, as indicated by the majority of feature scores

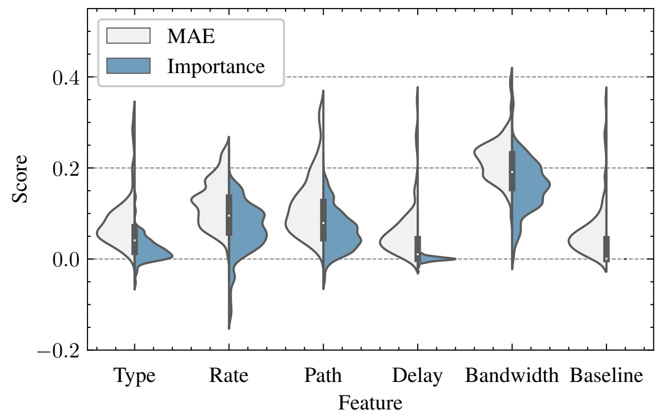


Fig. 14: MAE and feature importance score of each input feature as determined by the permutation-based MAE difference to the baseline. An unseeded random permutation is repeated 100 times to increase confidence.

being larger than zero. One outlier is the flow rate feature, which indicates that results derived from the model based on the rate lead to wrong results in a few cases. Therefore, feature engineering should be improved in the future to remove the flow rate anomalies. Second order feature importance is shown in Appendix A Figure 15.

V. Limitations

The current approach has a few limitations that are discussed in the following. The MARE is relatively high compared to state-of-the-art models for mean latency prediction. However, this is to be expected since taking the mean over a timeseries removes certain effects that need to be predicted correctly when modeling each timestep. For example, the build-up and draining of queues are reduced to a single value. The ability to apply this approach to networks with a number of nodes in the order of hundreds is limited because of the prohibitive cost of generating a training dataset. Some prediction artifacts remain with this dynamic graph encoding approach. The training time of such as temporal model is relatively high ($\times 36$) compared to a static model. The inference time per topology varies between 32 s and 47 s.

VI. Reproducibility

We provide access[6] to the three datasets as well as training and evaluation scripts.

VII. Conclusion

We implemented a data generation process that utilizes a network simulator to export sanitized queue utilization levels in networks with UDP flows competing for resources at multiple multiplexing points. We have shown that T-GNN models can capture spatio-temporal dependencies, such as queue build-up and draining phases with an error of 5.5%. The spatial component is required to accurately model flow paths, while the temporal component is needed to accurately model queue utilization over time. We have shown that the

[6] https://github.com/tgnn-test/dataset

temporal extension to GNNs outperforms the static approach during dynamic events in the network. Taking a permutation-based feature importance approach, we have shown that the T-GNN learns the correct dependencies between input features and latency behavior over time at each queue in the network.

References

[1] A. K. Erlang, "The theory of probabilities and telephone conversations," *Nyt. Tidsskr. Mat. Ser. B*, vol. 20, pp. 33–39, 1909.

[2] R. L. Cruz, "A Calculus for Network Delay. I. Network Elements in Isolation," *IEEE Transactions on information theory*, vol. 37, no. 1, pp. 114–131, 1991.

[3] J.-Y. Le Boudec and P. Thiran, *Network Calculus: a Theory of Deterministic Queuing Systems for the Internet*. Springer Science & Business Media, 2001, vol. 2050.

[4] Y. Jiang, Y. Liu *et al.*, *Stochastic Network Calculus*. Springer, 2008, vol. 1.

[5] S. Coles *et al.*, *An Introduction to Statistical Modeling of Extreme Values*. Springer, 2001, vol. 208.

[6] M. Ferriol-Galmés, K. Rusek, J. Suárez-Varela, S. Xiao, X. Shi, X. Cheng, B. Wu, P. Barlet-Ros, and A. Cabellos-Aparicio, "Routenet-Erlang: A Graph Neural Network for Network Performance Evaluation," in *IEEE INFOCOM*. IEEE, 2022, pp. 2018–2027.

[7] M. Peuster, H. Karl, and S. van Rossem, "MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016, pp. 148–153.

[8] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The Graph Neural Network Model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[9] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovic, "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges," *CoRR*, vol. abs/2104.13478, 2021. [Online]. Available: https://arxiv.org/abs/2104.13478

[10] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How Powerful are Graph Neural Networks?" *CoRR*, vol. abs/1810.00826, 2018.

[11] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal Graph Networks for Deep Learning on Dynamic Graphs," *arXiv preprint arXiv:2006.10637*, 2020.

[12] B. Rozemberczki, P. Scherer, Y. He, G. Panagopoulos, A. Riedel, M. Astefanoaei, O. Kiss, F. Beres, G. López, N. Collignon *et al.*, "Pytorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 4564–4573.

[13] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio, "Routenet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2260–2270, 2020.

[14] M. Ferriol-Galms, J. Paillisse, J. Surez-Varela, K. Rusek, S. Xiao, X. Shi, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "RouteNet-Fermi: Network Modeling with Graph Neural Networks," 2022.

[15] L. Bai, L. Yao, C. Li, X. Wang, and C. Wang, "Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting," 2020. [Online]. Available: https://arxiv.org/abs/2007.02842

[16] C. Zheng, X. Fan, C. Wang, and J. Qi, "GMAN: A Graph Multi-Attention Network for Traffic Prediction," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 01, 2020, pp. 1234–1241.

[17] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 922–929.

[18] J. Bai, J. Zhu, Y. Song, L. Zhao, Z. Hou, R. Du, and H. Li, "A3T-GCN: Attention Temporal Graph Convolutional Network for Traffic Forecasting," *ISPRS International Journal of Geo-Information*, vol. 10, no. 7, p. 485, 2021.

[19] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-GCN: A Temporal Graph Convolutional Network for Traffic Prediction," *IEEE transactions on intelligent transportation systems*, vol. 21, no. 9, pp. 3848–3858, 2019.

[20] B. Yu, H. Yin, and Z. Zhu, "Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting," *arXiv preprint arXiv:1709.04875*, 2017.

[21] Z. Yao, Q. Xu, Y. Chen, Y. Tu, H. Zhang, and Y. Chen, "Internet Traffic Forecasting using Temporal-Topological Graph Convolutional Networks," in *2021 IJCNN*. IEEE, 2021, pp. 1–8.

[22] Z. Wang, J. Hu, G. Min, Z. Zhao, Z. Chang, and Z. Wang, "Spatial-Temporal Cellular Traffic Prediction for 5 G and Beyond: A Graph Neural Networks-Based Approach," *IEEE Transactions on Industrial Informatics*, 2022.

[23] A. Taheri and T. Berger-Wolf, "Predictive Temporal Embedding of Dynamic Graphs," in *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2019, pp. 57–64.

[24] M. Wang, L. Hui, Y. Cui, R. Liang, and Z. Liu, "xNet: Improving Expressiveness and Granularity for Network Modeling with Graph Neural Networks," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 2028–2037.

[25] F. Geyer and S. Bondorf, "DeepTMA: Predicting Effective Contention Models for Network Calculus using Graph Neural Networks," ser. INFOCOM 2019, Paris, France, Apr. 2019.

[26] B. Jaeger, M. Helm, L. Schwegmann, and G. Carle, "Modeling TCP Performance Using Graph Neural Networks," ser. GNNet '22. New York, NY, USA: Association for Computing Machinery, Dec. 2022.

[27] S. Bondorf, "Worst-Case Performance Analysis of Feed-Forward Networks - An Efficient and Accurate Network Calculus," doctoralthesis, Technische Universität Kaiserslautern, 2016.

[28] C. D. Barros, M. R. Mendonça, A. B. Vieira, and A. Ziviani, "A Survey on Embedding Dynamic Graphs," *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, pp. 1–37, 2021.

[29] G. F. Riley and T. R. Henderson, "The ns-3 Network Simulator," in *Modeling and tools for network simulation*. Springer, 2010, pp. 15–34.

[30] B. K. de Aquino Afonso and L. Berton, "QT-Routenet: Improved GNN generalization to larger 5G networks by fine-tuning predictions from queueing theory," *ITU Journal on Future and Evolving Technologies*, vol. 3, no. 2, pp. 134–141, jul 2022.

[31] V. R. Joseph, "Optimal Ratio for Data Splitting," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 15, no. 4, pp. 531–538, 2022.

[32] A. Fisher, C. Rudin, and F. Dominici, "All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously," 2018. [Online]. Available: https://arxiv.org/abs/1801.01489

Appendix

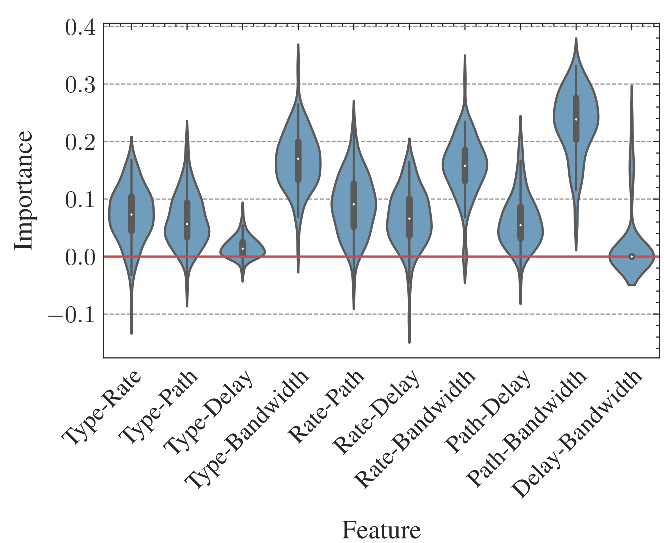


Fig. 15: Second-order feature importance of each combination of two input features