Sim2HW: Modeling Latency Offset Between Network Simulations and Hardware Measurements

Johannes Späth
spaethj@net.in.tum.de
Technical University of Munich
Chair of Network Architectures and Services
Munich, Germany

Max Helm
helm@net.in.tum.de
Technical University of Munich
Chair of Network Architectures and Services
Munich, Germany

Benedikt Jaeger
jaeger@net.in.tum.de
Technical University of Munich
Chair of Network Architectures and Services
Munich, Germany

Georg Carle
carle@net.in.tum.de
Technical University of Munich
Chair of Network Architectures and Services
Munich, Germany

Abstract

Network modeling often relies on simulation tools due to their flexibility and cost-effectiveness. However, in many cases, those tools can only cover some aspects of real-world networks accurately. Measurements on hardware testbeds are more accurate but require more resources and configuration and are thus frequently impractical for real-world networks. Graph Neural Networks (GNNs) are a promising machine learning approach proven to be especially useful for learning the properties of computer networks. In this paper, we present a GNN-based approach that uses simulation data as an additional input to predict latency values measured on real hardware. We train our model with an existing dataset from a hardware testbed and show that it can predict the latency distribution in unseen topologies with a MAPE of 27.2 % and an MdAPE of 19.8 %.

CCS Concepts

• **Networks** → **Network performance modeling**; *Network measurement*; *Network simulations.*

Keywords

Graph Neural Network; Latency Model; Network Simulation; Hardware Measurement

ACM Reference Format:
Johannes Späth, Max Helm, Benedikt Jaeger, and Georg Carle. 2024. Sim2HW: Modeling Latency Offset Between Network Simulations and Hardware Measurements. In *Proceedings of the 3rd GNNet Workshop: Graph Neural Networking Workshop (GNNet '24), December 9–12, 2024, Los Angeles, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3694811.3697820

GNNet '24, December 9–12, 2024, Los Angeles, CA, USA.

ACM ISBN 979-8-4007-1254-8/24/12
https://doi.org/10.1145/3694811.3697820

1 Introduction

Network modeling is a critical aspect of designing and optimizing modern communication systems. It enables researchers and operators to understand, predict, and enhance network performance without the need for costly and time-consuming real-world experiments. Moreover, network modeling is essential for identifying potential bottlenecks and failure points, allowing for proactive measures to ensure robustness and reliability. There exist multiple approaches to network modeling. Simulation tools are often used for their clear advantages, including cost-effectiveness, flexibility, and repeatability. However, simulations often fall short of predicting the behavior of real-world networks. This is due to abstractions in modeling the network stack and ignoring the influence of stochastic processes, such as, for example, interrupts. Some efforts have been made to address these types of shortcomings. For example, the network simulator ns-3 [12] has implemented direct code execution, allowing to replace elements of the simulated network stack with their counterpart implementations from the Linux kernel.[1] However, for precise simulations, many aspects need to be considered and included, which leads to a complex simulation design.

We aim to tackle the problem from a different angle. In this paper, we propose an approach that is based on Graph Neural Networks (GNNs) and uses data from a simple simulation as an additional input. That way, imperfections that arise from abstractions during simulation can be learned and corrected by the GNN and do not need to be simulated explicitly. We show that including simulation data has a positive effect on prediction accuracy. Throughout this work, we focus on modeling end-to-end UDP flow latencies in FIFO networks.

Our main contributions are:

(1) We propose an approach that uses data from simulations to predict latency values measured on hardware.
(2) We apply our model to an existing dataset obtained from close-to-hardware measurements in a testbed.
(3) We evaluate the strengths and weaknesses of the approach, taking into account different percentiles of the predicted latency distributions and different network properties.

[1] https://www.nsnam.org/docs/dce/release/1.1/manual/singlehtml/, visited on 2024-10-05.

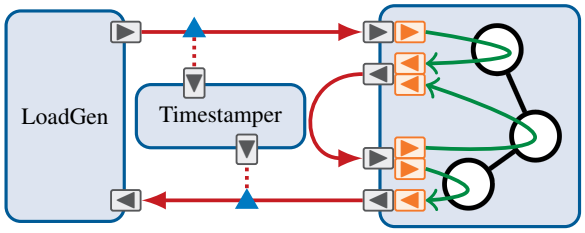


Figure 1: The measurement setup for obtaining the HVNet dataset [18], Figure from Helm and Carle [7]. Physical cables and interfaces are denoted in red and grey, respectively. Within the hypervisor, virtual functions (orange) are used to implement the connections (green) between the VMs.

The remainder of the paper is structured as follows. Section 2 provides an overview of the background, including relevant information on the hardware measurements, the simulation framework OMNeT++, and GNNs. Section 3 reviews related work in the area of network performance prediction using GNNs. In Section 4, we describe our approach in detail. Section 5 presents and discusses our results. Finally, we conclude our contributions in Section 6.

2 Background

This section provides information on hardware measurements, discrete event simulations, and GNNs.

2.1 Hardware Measurements

Since we want to predict the latency distribution of flows in real-world networks, we require realistic flow data for training and evaluation. Obtaining such data can be costly, especially when setting up large topologies on real hardware. Using virtualization helps to reduce costs significantly. However, this usually comes at the price of reduced data accuracy. Wiedner et al. [17] presented HVNet, a framework that uses Virtual Machines (VMs) but stays close to real hardware with respect to networking. They achieved this by using Single Root I/O Virtualization (SR-IOV) and sending packets over an actual physical link instead of virtualized networking.

The authors provide a dataset [18] of end-to-end flow latency measurements for 100 random topologies. Their measurement setup for obtaining this data is depicted in Figure 1. Three hardware nodes are involved, a load generator (*LoadGen*), a time-stamping device (*Timestamper*), and a hypervisor running the VMs for the network nodes. The load generator produces the traffic for each flow, sends it over the physical wire to the hypervisor, and injects it into the VM at which the flow originates. This is done by using SR-IOV and different virtual functions at the hypervisor. The traffic traverses the virtualized network using a physical loop wire, again utilizing SR-IOV and multiple virtual functions. That way, it is ensured that the packets traverse physical networking equipment for each hop. The time stamper is connected to the links of the load generator using an optical splitter, allowing for precise end-to-end latency measurements. We use the HVNet dataset for training and evaluating our model, and we describe its properties in more detail in Section 4.2.

2.2 Discrete Event Simulations

Another option to generate network datasets is to use simulation tools, which is significantly cheaper than conducting measurements on real hardware. In that regard, various strategies exist, which mainly differ in the simulation target, such as packets, flows, or events. The latter approach is also known as *discrete event simulation*, with popular tools implementing it being OMNeT++ [15] and ns-3 [12]. The OMNeT++ simulator offers a modular design and can be used for various simulation tasks. Computer network simulations can be performed using the INET framework [10], which offers models for common protocols of the Internet stack. We use OMNeT++ and INET to replicate the topologies in the HVNet dataset and obtain simulated latency values that we use as additional input to our GNN model.

2.3 Graph Neural Networks

GNNs are a geometric machine learning approach working directly on graphs as inputs. They are permutation-invariant with respect to the input graphs, that is, isomorphic graphs are treated equally independent of the encoding. The input to a GNN is a graph consisting of nodes and edges $G = (V, E)$ with attached node and edge features. The graph is encoded as an adjacency matrix and a node feature matrix. During training and inference, the states of the nodes are updated based on the states of their neighboring nodes. GNNs can be roughly divided into three partially overlapping categories: convolutional, attention, and message-passing [2]. We utilize a convolutional-like variant [6]. At an abstracted level, updating node states in the convolutional variant works as shown in Equation (1) [2], where $\bigoplus$ is a permutation-invariant operation, for example, sum or mean.

$$h_u = \phi\left(x_u, \bigoplus_{v \in N_u} c_{uv}\psi(x_v)\right) \tag{1}$$

The function ϕ updates the node state h_u using the current state of the node and a combination of node states of the nodes v in the neighborhood N_u. The function ψ is an optional transformation on the node state x_v. Finally, c_{uv} is a weight factor. [2]

3 Related Work

Rusek et al. [13] presented a GNN architecture for modeling computer networks. Their model predicts the per-packet delay distribution, however, they just employ simulated data for this task and merely use a total of 4 distinct topologies.

Ferriol-Galmés et al. [4] proposed a Digital Twin model based on GNNs that they used for predicting per-path delays. They used samples from only two topologies during training and could accurately predict samples for 106 unseen topologies from the Internet Topology Zoo [8]. For this part, the authors still just used data from a simulator, but they also evaluated their model in a further step by utilizing data from a hardware testbed. The prediction target of their model focuses on the mean latency, not taking into account the distribution of the values.

Wang et al. [16] used a more granular prediction target that focuses on the mean values at every time step instead of just the overall mean. They achieve this by employing a factorization-based

Work	Ref.	Year	Data Source	Train Topologies	Unseen Test Topologies	Max. Network Size	Prediction Target
Rusek et al.	[13]	2020	*Sim*	3	1	50	Normal, (gamma) distribution
Ferriol-Galmés et al.	[4]	2022	*Sim*, *HW*	2 (*Sim*)	106 (*Sim*)	95 (*Sim*) / 8 (*HW*)	Mean
Wang et al.	[16]	2022	*Sim*	2	1	24	Mean per time step
Yang et al.	[19]	2022	*Sim*	9	0?	>128	Distribution (mean, p_{99} reported)
Güemes-Palau et al.	[5]	2023	*HW*	≤11	<11	8	Mean
Helm and Carle	[7]	2023	*HW*	87*	10*	15	Mean + Percentiles
This Work	—	2024	*Sim*, *HW*	71 (*Sim* + *HW*)	18 (*Sim* + *HW*)	15 (*Sim* + *HW*)	Percentiles

Table 1: An overview of related work in network performance prediction using GNNs. *Sim* refers to data obtained via simulation, *HW* to data gained by measurements on hardware testbeds.

temporal methodology. Yang et al. [19] adapted the target by predicting a latency distribution, and they report the mean and 99^{th} percentile. Both approaches, however, only utilize data from a simulator and investigate a relatively small number of distinct topologies.

Güemes-Palau et al. [5] presented a solution to the GNN Challenge 2023 [14]. The challenge's goal was to build a neural network architecture that accurately predicts the mean per-flow latency with data from hardware measurements. For that, a dataset with traffic from a hardware testbed was provided, consisting of data from 11 unique topologies of up to 8 nodes. The proposed solution focuses on optimizing the *RouteNet-Fermi* GNN architecture introduced by Ferriol-Galmés et al. [3] and achieves a Mean Absolute Percentage Error (MAPE) of 27.8 % with their test dataset.

Helm and Carle [7] suggested an approach for predicting latency percentiles in the dataset by Wiedner et al. [18]. They included bounds derived from the Network Calculus (NC) framework to correct prediction errors, which they showed to be especially helpful with regard to higher percentiles. Note that they used the same *Hardware Dataset* as we, however, they shuffled the topologies used for training and evaluation multiple times to cross-validate their results.

Table 1 gives an overview of related work.

Our approach differs from the work proposed by others in multiple aspects. First, we use data from close-to-hardware measurements and evaluate a relatively large number of unique topologies during training and testing. Moreover, our model includes data gathered by replicating the input network in a discrete event simulator to improve prediction accuracy. Further, our prediction target comprises multiple percentiles of the latency distribution instead of just the mean. Finally, we do not optimize the GNN model besides Hyper-Parameter Optimization (HPO).

4 Methodology

This section describes the methodology divided into architecture, datasets, and training process.

4.1 Architecture

An overview of our approach is depicted in Figure 2. As a basis, we use the 100 topology and flow definitions from the HVNet dataset [18]. Wiedner et al. used the HVNet framework [17] to create close-to-hardware measurements for those definitions, which we utilize as our *Hardware Dataset*. We generate the *Simulation Dataset* by converting the same definitions to OMNeT++ configuration files and collecting the result data of the simulation. We thus end up with two datasets with latency data for the same topologies

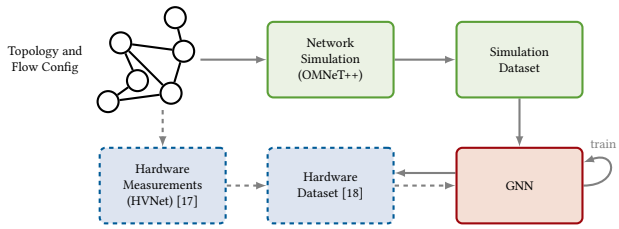


Figure 2: Overview of our approach. We utilize OMNeT++ to simulate the networks in the HVNet dataset [18] and train a GNN to predict the data measured from hardware using simulated data as an input.

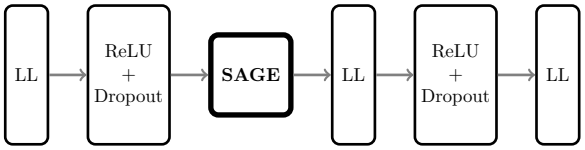


Figure 3: Overview of our GNN architecture, consisting of multiple Linear Layers (LLs), Rectified Linear Units (ReLUs), Dropout Layers (Dropouts), and a Sample and Aggregate (SAGE) [6] operator.

and flows, one generated by measurements and one by simulations. Both datasets' network and flow properties are described in more detail in Section 4.2.

For a more accurate simulation, we model the sending behavior of the flows in the *Hardware Dataset* by fitting a gamma distribution over their inter-send times, and we configure OMNeT++ to simulate sending following the same distribution. This is required, as the original distribution that was used to generate the *Hardware Dataset* is not available. We model sending per distinct flow rate, that is, flows of the same rate follow the same gamma distribution.

We characterize the distribution of flow latency values by different percentiles, starting from the *minimum* (that is, p_0) up to $p_{99.999}$. We do not include the *maximum*, as the distribution up to $p_{99.999}$ should be accurate enough to model the latency behavior of flows.

Then, we convert the network topology, flow properties, and latency percentiles from both the *Hardware* and *Simulation Dataset* to a graph representation that we use as input for the GNN. Numerical values, such as the latency values, are normalized using the logarithmic function $log(x + 1)$, categorical values are encoded as one-hot.

As our main focus lies on the effect of simulation data on prediction accuracy of our model, we do not want the GNN architecture

Metric	Min.	Max.
Network size	8	15
Number of flows	19	59
Flow length	2	9
Flow rate	1 Mbit/s	831 Mbit/s
Max. link util. per flow	0.11 %	87 %

Table 2: Metrics of the datasets.

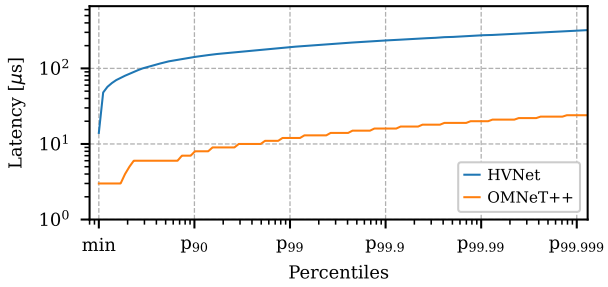


Figure 4: High Dynamic Range (HDR) plot of end-to-end flow latency values observed for an exemplary topology in both the *Hardware* and *Simulation Dataset*.

to have a big influence. Thus, we use a simple default architecture, which is depicted in Figure 3. The input data first traverses a LL, a ReLU, and a Dropout, before being passed to a SAGE [6] operator. Afterwards, it is forwarded through a LL, a ReLU, a Dropout, as well as another LL. We design the GNN to predict the latency percentiles of the *Hardware Dataset* per flow. Thus, we use the latency distribution obtained via simulation as an input to predict the latency distribution that would have been measured in a hardware setup.

4.2 Hardware and Simulation Datasets

The *Hardware Dataset* [18] that we use as a basis comprises 89 unique topologies[2] and a total of 3191 flow definitions. In this context, a flow refers to a sequence of UDP packets traversing a fixed network path. Table 2 lists some metrics of the dataset. These properties also apply to the *Simulation Dataset*, as we use the same topology and flow configurations for the simulations.

When comparing the latency distribution of an exemplary topology in both the *Hardware* and *Simulation Dataset* (see Figure 4), we observe that the latency values measured by HVNet are approximately one order of magnitude larger than the ones simulated by OMNeT++. Further, the simulated data has visible plateaus, that is, groups of samples that share the same latency values, which we do not see in the measured data. Another clear difference is the behavior for small percentiles, where the values measured by HVNet follow a much steeper curve than the ones simulated by OMNeT++. Apart from that, the latency percentiles show a similar trend across both datasets. The main goal of our model is to learn the offset between simulation and measurement data.

[2] Originally, 100 topologies were defined by Wiedner et al., however, not all measurements were successful, resulting in the reduced number of 89 topologies.

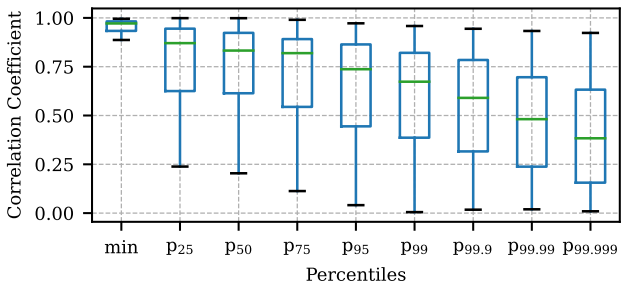


Figure 5: Pearson correlation coefficient between the different topologies in the HVNet and OMNeT++ datasets, regarding different statistic measures. The boxes of the plot show the interquartile range (IQR) between Q_1 and Q_3, the whiskers denote the furthest data point lying within the 1.5-fold IQR.

To quantify the relationship between the two datasets, we calculate the Pearson correlation coefficient between the latency percentiles taken from both the *Hardware* and *Simulation Dataset* for each topology. The results are denoted in Figure 5. We note a clear relationship between those values, which gives a good indication that a GNN might be able to learn this correlation and thus motivates our approach to include the simulation data in the model. Especially for lower percentiles (from the *minimum* up to p_{75}), the Pearson coefficient is high, degrading significantly for higher percentiles. Thus, we assume that a major challenge are latency outliers that exist in the *Hardware Dataset* but are not modeled by the simulation.

4.3 Training and Hyper-Parameter Optimization

A primary challenge during the training process is the relatively small dataset, since we only have a total of 3191 flow definitions. We randomly split the 89 distinct topologies into a training set of 71 topologies (2506 flows) and an evaluation set of 18 topologies (685 flows), roughly following an 80:20 split. The topologies in both sets are comparable with respect to the metrics described in Table 2. Due to the limited size of the dataset and the low correlation between the simulated and measured latency values for high percentiles (see Section 4.2), we do not expect our model to achieve prediction errors that are comparable to related work that is predicting simulated latency values.

For tuning the hyper-parameters, we use the Neural Network Intelligence (NNI) tool [11]. Its HPO feature offers a convenient way to programmatically explore a given search space for multiple parameters and find an optimal combination. The result parameters discovered during this process, which we use for our model, are denoted in Appendix A.

As a loss function, we selected the Mean Squared Error Loss (MSELoss).

5 Evaluation

In this section, we present and discuss our results. Note that we base all evaluations on the model from training epoch 90, which

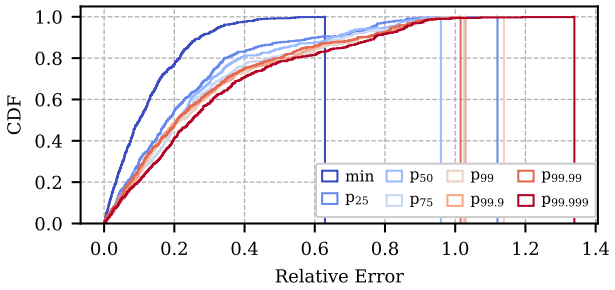


Figure 6: CDF plot showing the distribution of the relative errors for different prediction targets.

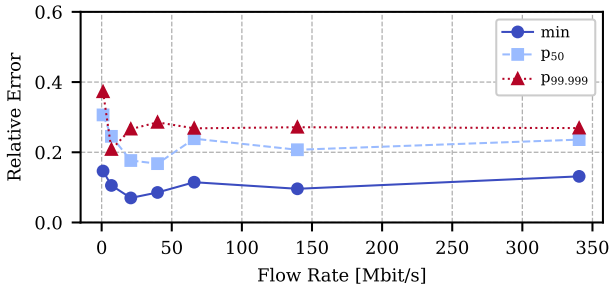


Figure 7: Relative error of predictions for different percentiles as they relate to the rate of the flows.

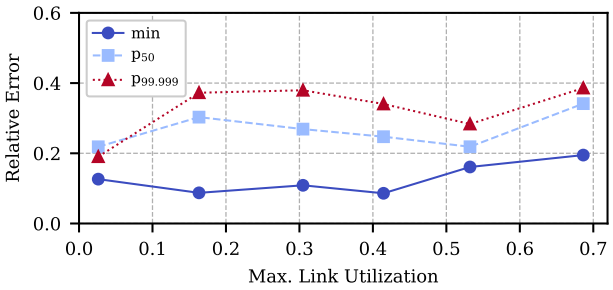


Figure 8: Relative error of predictions for different percentiles as they relate to the maximum link utilization on the flow paths.

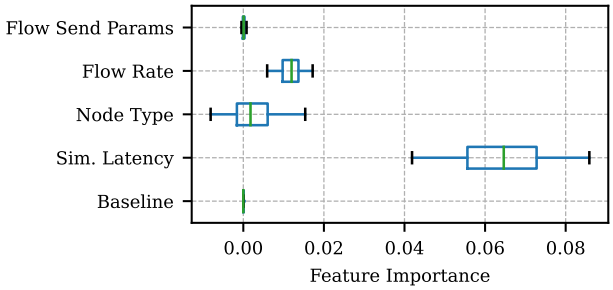


Figure 9: Importance of different input features of the GNN. A higher value means that the feature is more important.

was performing best concerning the MAPE. Further, note that if not specified differently, the boxes of all box plots show the IQR between Q_1 and Q_3, and the whiskers denote the furthest data point that lies within the 1.5-fold IQR. Moreover, if not stated differently, we utilize the MAPE and the relative error as metrics for our model's performance. We define the latter as $|\frac{y'}{y} - 1|$, with y being the actual value and y' being the prediction.

5.1 Prediction Accuracy

When testing our model with data from the 18 topologies in the evaluation dataset (see Section 4.3), we achieve a MAPE of 27.2 % and an Median Absolute Percentage Error (MdAPE) of 19.8 % over all prediction targets. To better understand our model's accuracy, we first look at the results for the prediction targets individually. Figure 6 shows the distribution of the relative error for all 685 flows regarding various percentiles as CDF. As the correlation coefficients in Figure 5 already indicated, the *minimum* prediction is the most accurate one, with a median relative error of 10.1 %. For higher percentiles, this value increases up to a median of 25.0 % for $p_{99.999}$. Furthermore, there exists a flat area that separates samples that we can predict well (relative error <40 %) and samples that our model fails to predict accurately (relative error >80 %). We investigate those outliers in more detail in Section 5.4.

5.2 Impact of Network Properties

To gain a better understanding of the effect of network characteristics on prediction accuracy, we investigate the error with respect to different properties of the data samples. Figure 7 shows the impact of the flow rate on the relative error for different latency percentiles. Further, Figure 8 depicts the effect of the maximum link utilization that a flow is experiencing along its path. For better visualization, we aggregate multiple flow rates and maximum link utilization values into only a few data points. This is done by splitting all data points into equally sized bins and calculating the mean of the values in each bin.

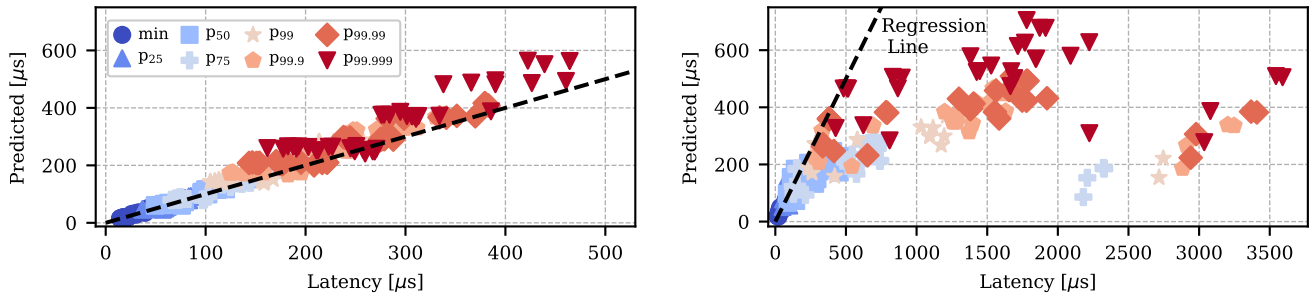
As shown above in Figure 6, the prediction of the *minimum* is the most accurate one. Additionally, the accuracy is decreasing for higher percentiles. We observe an increased error for very low flow rates, however, besides that, we see no significant impact of the flow rate on prediction accuracy. The dataset includes many small flows and only a few high-rate ones, which can be deduced from the high frequency of data points for lower rates. Therefore, we do not have enough data about high-rate flows to make meaningful statements on how accurate our model can predict those.

For increasing values of maximum link utilization, we can observe a decreased prediction accuracy across all considered target percentiles. Especially for $p_{99.999}$, the relative error significantly increases from 19.0 % (max. util. ~3 %) to 38.6 % (max. util. ~69 %).

We conclude that the latency of flows traversing highly utilized links is harder to predict with our approach.

5.3 Feature Importance

In a further step, we aim to quantify the impact of different input features on prediction accuracy. For that, we randomly permute all values of the respective feature in the test dataset, let the model predict, and compare the effect on the MAPE to the baseline with



(a) Topology with a high prediction accuracy (13.6 % MAPE).

(b) Topology with a low prediction accuracy (47.9 % MAPE).

Figure 10: Measured and predicted latency percentiles for two topologies from the test dataset, one achieving high accuracy, the other one suffering from flows with very high latency values that our model cannot predict accurately. The regression line shows the optimal position of the data points, where the predictions exactly match the measured latency percentiles.

non-permuted values. We aggregate the percentiles of the simulated latency values to a single feature here, as we are mostly interested in the importance of adding simulation data and not in the effect of the individual percentiles. Figure 9 shows the results of this process for different input features. We can clearly see that the simulated latency percentiles have the biggest positive influence on the predictions. This shows that including simulation data into our model improves prediction accuracy. Regarding the flow properties, the sending behavior has almost no influence, while the flow rate is more important. The node type also has a visible effect on the predictions, as it plays an important role in assigning meaning to the structural elements of the graphs.

5.4 Hardware Dataset Mispredictions

As mentioned in Section 5.1, the *Hardware Dataset* contains samples that our model is not able to predict accurately. We compare the predictions for two topologies to further evaluate the reason for this. Regarding their network and flow properties, no significant difference can be noted. However, for the first one (Figure 10a), our model achieves an accuracy of 13.6 % MAPE, while for the second one (Figure 10b), the MAPE is as high as 47.9 %. The reason for this low accuracy are a few "outlier flows" with latency values more than ten times larger than most of the other flows, which can be seen in the three clusters on the right side of the regression line in Figure 10b. As this behavior can neither be inferred from the input features, nor is it covered by the simulator, our GNN cannot learn the offset between simulation and measurements in these cases. Those outlier flows also explain the flat area between 40 % and 80 % MAPE in Figure 6.

On the other hand, Figure 10a shows that without these effects, our approach can accurately predict hardware-measured latency values.

5.5 Negative Results

Using different normalization functions such as *min-max* or *z-score* normalization did not result in a higher prediction accuracy. Furthermore, we weighted the loss individually for the distinct prediction targets (that is, the latency percentiles). However, this approach did also not prove helpful. In addition, we tried a Graph Attention Network v2 (GATv2) [1] and Gated Graph Sequence Neural Network (GGNN) [9] instead of the SAGE [6] layer in our GNN architecture (see Figure 3), but this also resulted in less accurate predictions.

6 Conclusions and Future Work

In this paper, we presented a GNN-based approach that uses data obtained via simulation to predict end-to-end flow latency values measured on a hardware testbed. We applied our model to an existing dataset obtained from testbed measurements and showed that it can generalize over different topologies, achieving a MAPE of 27.2 % and an MdAPE of 19.8 % over all prediction targets. We also demonstrated that with our approach, the *minimum* and lower percentiles are easier to predict than percentiles close to the *maximum*. Furthermore, a few outlier flows with unexpectedly high latency values exist in some topologies of the *Hardware Dataset*, which cannot be learned by the GNN due to the absence of those effects in both the simulation data and the input features. Further, the small size of the dataset limits our model and prevents it from making more accurate predictions.

Future work should address this problem and also utilize a more diverse *Hardware Dataset* that is not limited to a stateless protocol (UDP). In this regard, other aspects like flow dynamics and routing policies could also be taken into account to validate the approach with more realistic data.

We provide access to our dataset, the GNN model, and other artifacts.[3]

Acknowledgments

This work was supported by the EU's Horizon 2020 programme as part of the projects SLICES-PP (10107977) and GreenDIGIT (4101131207), by the German Federal Ministry of Education and Research (BMBF) under the projects 6G-life (16KISK002) and 6G-ANNA (16KISK107), and by the German Research Foundation (HyperNIC, CA595/13-1).

[3] https://github.com/tumi8/sim2hw-gnnet-paper, visited on 2024-10-05

References

[1] Shaked Brody, Uri Alon, and Eran Yahav. 2021. How Attentive are Graph Attention Networks? *CoRR* abs/2105.14491 (2021). arXiv:2105.14491 https://arxiv.org/abs/2105.14491

[2] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Velickovic. 2021. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *CoRR* abs/2104.13478 (2021). arXiv:2104.13478 https://arxiv.org/abs/2104.13478

[3] Miquel Ferriol-Galmés, Jordi Paillisse, José Suárez-Varela, Krzysztof Rusek, Shihan Xiao, Xiang Shi, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2023. RouteNet-Fermi: Network Modeling With Graph Neural Networks. *IEEE/ACM Transactions on Networking* 31, 6 (2023), 3080–3095. https://doi.org/10.1109/TNET.2023.3269983

[4] Miquel Ferriol-Galmés, José Suárez-Varela, Jordi Paillissé, Xiang Shi, Shihan Xiao, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2022. Building a Digital Twin for Network Optimization Using Graph Neural Networks. *Computer Networks* 217 (2022), 109329. https://doi.org/10.1016/j.comnet.2022.109329

[5] Carlos Güemes-Palau, Miquel Ferriol Galmés, Albert Cabellos-Aparicio, and Pere Barlet-Ros. 2023. Building a Graph-Based Deep Learning Network Model From Captured Traffic Traces. arXiv:2310.11889 [cs.NI] https://arxiv.org/abs/2310.11889

[6] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. *Advances in Neural Information Processing Systems* 30 (2017).

[7] Max Helm and Georg Carle. 2023. Predicting Latency Quantiles Using Network Calculus-Assisted GNNs. In *Proceedings of the 2nd on Graph Neural Networking Workshop 2023*. Association for Computing Machinery, Paris, France, 13–18. https://doi.org/10.1145/3630049.3630173

[8] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. 2011. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications* 29, 9 (2011), 1765–1775.

[9] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2017. Gated Graph Sequence Neural Networks. arXiv:1511.05493 [cs.LG] https://arxiv.org/abs/1511.05493

[10] Levente Mészáros, Andras Varga, and Michael Kirsche. 2019. INET Framework. *Recent Advances in Network Simulation: The OMNeT++ Environment and its Ecosystem* (2019), 55–106.

[11] Microsoft. 2021. *Neural Network Intelligence*. https://github.com/microsoft/nni

[12] George F. Riley and Thomas R. Henderson. 2010. *The ns-3 Network Simulator*. Springer Berlin Heidelberg, Berlin, Heidelberg, 15–34. https://doi.org/10.1007/978-3-642-12331-3_2

[13] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2020. RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN. *IEEE Journal on Selected Areas in Communications* 38, 10 (2020), 2260–2270.

[14] José Suárez-Varela et al. 2021. The Graph Neural Networking Challenge: A World-Wide Competition for Education in AI/ML for Networks. *ACM SIGCOMM Computer Communication Review* 51, 3 (2021), 9–16.

[15] András Varga and Rudolf Hornig. 2010. An Overview of the OMNeT++ Simulation Environment. ICST. https://doi.org/10.4108/ICST.SIMUTOOLS2008.3027

[16] Mowei Wang, Linbo Hui, Yong Cui, Ru Liang, and Zhenhua Liu. 2022. xNet: Improving Expressiveness and Granularity for Network Modeling with Graph Neural Networks. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. 2028–2037. https://doi.org/10.1109/INFOCOM48880.2022.9796726

[17] Florian Wiedner, Max Helm, Sebastian Gallenmüller, and Georg Carle. 2022. HVNet: Hardware-Assisted Virtual Networking on a Single Physical Host. In *IEEE INFOCOM WKSHPS: Computer and Networking Experimental Research using Testbeds (CNERT 2022) (INFOCOM WKSHPS CNERT 2022)*. Virtual Event. https://doi.org/10.1109/INFOCOMWKSHPS54753.2022.9798351

[18] Florian Wiedner, Max Helm, Sebastian Gallenmüller, and Georg Carle. 2022. HVNet: Hardware-Assisted Virtual Networking on a Single Physical Host. https://mediatum.ub.tum.de/1638129

[19] Qingqing Yang, Xi Peng, Li Chen, Libin Liu, Jingze Zhang, Hong Xu, Baochun Li, and Gong Zhang. 2022. DeepQueueNet: Towards Scalable and Generalized Network Performance Estimation with Packet-Level Visibility. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 441–457.

A Appendix

Parameter	Value
Batch size	2
Dropout Linear*	0.000173
Dropout GRU*	0.228
Hidden size*	32
Learning rate*	0.00541
Loss function	MSELoss
LR scheduler	`ReduceLROnPlateau`
LR scheduler factor*	0.7
Number of loop unrolling for SAGE*	4
Train-test split	0.85

Table 3: Hyper-parameters used for training the model. Parameters marked with an asterisk (*) were determined using NNI's HPO.