UNIVERSITÄT AUGSBURG
Universitätsbibliothek

OPEN ACCESS
OPUS AUGSBURG

ML-based performance prediction of SDN using simulated data from real and synthetic networks

Katharina Dietz, Nicholas Gray, Michael Seufert, Tobias Hoßfeld

Angaben zur Veröffentlichung / Publication details:

Dietz, Katharina, Nicholas Gray, Michael Seufert, and Tobias Hoßfeld. 2022. "ML-based performance prediction of SDN using simulated data from real and synthetic networks." In *NOMS 2022 - 2022 IEEE/IFIP Network Operations and Management Symposium, April 25-29, 2022, Budapest, Hungary*, edited by Pal Varga, Lisandro Zambenedetti Granville, Alex Galis, Istvan Godor, Noura Limam, Prosper Chemouil, Jérôme François, and Marc-Oliver Pahl, 1–7. Piscataway, NJ: IEEE. https://doi.org/10.1109/noms54207.2022.9789916.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
Deutsches Urheberrecht
Weitere Informationen finden Sie unter: / For more information see:
https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/

ML-based Performance Prediction of SDN using Simulated Data from Real and Synthetic Networks

Katharina Dietz, Nicholas Gray, Michael Seufert, Tobias Hossfeld
University of Würzburg, Germany
{katharina.dietz, nicholas.gray, michael.seufert, tobias.hossfeld}@uni-wuerzburg.de

***Abstract*—With increasing digitization and the emergence of the Internet of Things, more and more devices communicate with each other, resulting in a drastic growth of communication networks. Consequently, managing these networks, too, becomes harder and harder. Thus, Software-defined Networking (SDN) is employed, simplifying the management and configuration of networks by introducing a central controlling entity, which makes the network programmable via software and ultimately more flexible. As the SDN controller may impose scalability and elasticity issues, distributed controller architectures are utilized to combat this potential performance bottleneck. However, these distributed architectures introduce the need for constant synchronization to keep a centralized network view, and controller instances need to be placed in appropriate locations. As a result, thoroughly designing SDN-enabled networks with respect to a multitude of performance metrics, e. g., latency and induced traffic, is a challenging task. To assist in this process, we train a performance prediction model based on properties which are available during the network planning phase. We utilize a simulation-based approach for data collection to cover a large parameter space, simulating a variety of networks and controller placements for two opposing SDN architectures. On basis of this dataset, we apply Machine Learning (ML) to solve the performance prediction as a regression problem.**

I. Introduction

The complexity of modern communication networks is rapidly increasing, caused by emerging trends such as Industry 4.0 and Internet of Things. Thus, Software-defined Networking (SDN) is incorporated into the network design to increase the manageability and flexibility of a communication system [1], [2]. By introducing SDN into a network, a single controller instance is now responsible for the management and configuration of the whole network, which may result in a performance bottleneck and a single point of failure. Therefore, distributed SDN architectures are a common approach to avoid this potential problem. These distributed architectures balance the network load among several physical controller instances, while still keeping the logical network view centralized. With distributed architectures, new challenges arise, however, most importantly the required synchronization of the network state [3] and the placement of the controller instances within the networks [4]. Consequently, SDN-enabled networks have to be planned carefully, including the overall layout of the network, the utilized architecture, and controller configuration.

In recent years, research has mainly focused on the evaluation of specific parts of the SDN ecosystem, such as benchmarking different controller implementations concerning throughput or latency. Other approaches utilize the convenience of the centralized network view to collect network measurements and evaluate the service quality and user experience of dedicated services. In many cases, the network is already deployed and the analysis focuses on a particular service or SDN component of a specific network configuration [5], [6].

In this work, we tackle the problem of predicting the performance of a network purely based on properties that are known before deployment, e. g., topological metrics or controller configurations. This prediction of operational performance can serve as a helpful tool for network design already in the network planning phase.

The first contribution of this work is the creation and publication[1] of a dataset with various dynamic performance and topological metrics of fifty simulated, real-world SDN-enabled networks for various configurations. Secondly, we compare multiple preprocessing steps to reduce the dimensionality of this dataset and evaluate the resulting performance prediction of different Machine Learning (ML) models. Lastly, we evaluate well-known graph generators to create synthetic data to increase the size of our dataset and to substitute the real-world networks during training.

The remainder of this work is structured as follows. Firstly, in Section II, we give background information about SDN, distributed architectures, and the OpenFlow OMNeT++ Suite[2] (OOS) which was used to simulate SDN-enabled networks. Section III outlines related work to our research topic. In Section IV, we specify the simulation setup, dataset description, and ML pipeline, followed by an evaluation of SDN architectures. We then evaluate the capability of well-known graph generators to serve as training data for the real-world networks in Section V. Lastly, we summarize our findings and provide an outlook for future work in Section VI.

II. Background

In this section, we briefly outline the main concept behind SDN, with the OpenFlow protocol being one of its pioneers. Furthermore, we present two different approaches for distributed architectures, as well as the OMNeT++ framework and the extension OpenFlow OMNeT++ Suite[2] (OOS).

[1]https://github.com/lsinfo3/noms2022-sdn-performance-prediction
[2]https://github.com/lsinfo3/OpenFlowOMNeTSuite

OpenFlow & Standard SDN Architecture. The core principle of SDN is the separation of a network's intelligence and its executive authority, i. e., the separation of control and data plane. The OpenFlow[3] [7] protocol is one of the first enablers to facilitate the communication between both planes by standardizing the interaction of the controller with the switches on the data plane. OpenFlow switches install flow tables, which match incoming packets and determine further actions, e. g., the next device the packet is sent to. The controller is able to manage these devices and their flow tables, such as configuring the lifetime of the entries in the table, also called timeout value. If the switch is able to handle the packet autonomously, this induces no traffic onto the controller and no delay until packet forwarding, called the fast path. If no entry matches the packet, the packet is relayed to the controller for further information, called the slow path. It is desirable to keep the relayed traffic at minimum, while keeping the complexity of flow tables low due to limited resources [8], [9].

Distributed Architectures & Controller Placement. As the single controller instance may impose a risk in case of overloading when too many switches are requesting responses from the controller, distributed architectures are employed. To balance the network load physically, different approaches can be pursued, i. e., *flat* and *hierarchical* approaches. One representative of a flat approach is HyperFlow [10], in which all instances maintain a global view and require constant synchronization. Kandoo [11] follows a hierarchical approach by introducing a root and several local controllers. The local instances only maintain a partial view and query the root if a task is out of their responsibility, e. g., forwarding a packet to a switch managed by another controller. However, not only do distributed architectures generate overhead in terms of synchronization, but also introduce the Controller Placement Problem (CPP), which describes how many controllers are needed and where to place them [12]. This is a multi-objective optimization problem with competing optimization goals, e. g., minimizing controller-to-switch latency (C2SL) and controller-to-controller latency (C2CL), or respecting balanced switch-to-controller mappings. Changing the placement of controllers may severely impact the performance of the networks, as this may completely change the partial network view of local controller instances. Thus, a careful evaluation of the interplay between the myriad of different influence factors on a network's performance is an important task.

OMNeT++ & OOS. Even though SDN is an emerging design paradigm for modern networks, access to large scale software-defined networks and compatible hardware is still limited [13], [14]. As a consequence, collecting data from a wide variety of networks, multiple distributed architectures, and controller placements or configurations is non-trivial. Thus, we follow a simulation-based approach by utilizing the OMNeT++ simulation framework[4]. We make use of the OOS, which not only provides all necessary tools to mimic the functionality the core OpenFlow components, but also altered versions for HyperFlow and Kandoo. Additionally, the OOS contains fifty real-world Wide Area Networks (WANs), ported from the Internet Topology Zoo[5] [15] and Internet2's Advanced Layer 2 Service[6] (AL2S).

III. Related Work

In this section, we give an overview of related work concerning our research to point out similarities and differences.

SDN Controller & Switch Performance. As showcased in [6], there exists a multitude of controller implementations, written in a variety of programming languages and constructed by diverging architectures. Thus, one important task is to benchmark different implementations regarding their performance, e. g., throughput and latency. For this, the authors in [16] propose *Cbench*, a popular tool for benchmarking already deployed devices. As access to deployed SDN-enabled networks is still limited, many works resort to either simulation-based approaches [17], [18], utilize emulators [19], [20], or mathematical models [9].

Just as important as an evaluation of the control plane, is an analysis of the data plane, i. e., the switches. As for controllers, there exists a variety of switches, some of which are evaluated in [21]. Consequently, similar benchmarking analyses regarding performance metrics may be conducted if the deployed network can be accessed for performance studies. As this is rarely possible, analogous to controller evaluations, some works leverage simulations [22], while others utilize mathematical models [23], [24].

In this work, we do not focus on a single SDN component, but rather the whole SDN ecosystem. While latency- and workload-based performance metrics are still important in our analyses, we also focus on the interaction between the SDN components, i. e., control and data plane.

ML-based Performance Prediction in SDN. The SDN controller receives numerous amounts of data and with the recent rise in popularity of big data, ML-based approaches offer opportunities to gain superior insights into the data compared to traditional approaches. In many cases, the centralized controller simplifies the data collection, which is then used as input for ML-based approaches in an offline manner. In other cases, the ML algorithm is directly deployed onto the controller to enable real-time decision making during runtime, e. g., for security purposes [25].

In terms of SDN-enabled performance prediction, many different goals can be pursued. On the network layer, performance and traffic forecasting is the basis for various tasks such as routing [26] or load balancing [27]. On the application layer and closer to end-users, the authors in [28] secure the Quality of Service (QoS) of a specific application, or even further, the works in [29], [30] ensure the Quality of Experience (QoE) of end-users. A more high-level work is [31], where the authors identify correlations between various Key Performance

[3]https://opennetworking.org/software-defined-standards/specifications/
[4]https://omnetpp.org/
[5]http://www.topology-zoo.org/
[6]https://internet2.edu/services/layer-2-service/

Indicators (KPIs) and Key Quality Indicators (KQIs) as means to enable root cause analysis for degraded QoS.

In the above works, the network is already online and the aim is to optimize the performance of a specific service or task. In this work, however, we neither focus on gaining benefits of a centralized controller, nor do we predict the performance of a specific service in an already deployed network. Rather, we evaluate the performance of the network as a whole. The core idea is to analyze its performance before the actual deployment to support the network design process, e. g., when choosing the underlying topology or controller architecture.

Compared to our initial work [32], we greatly increased the size of our dataset by including a variety of controller placements and configurations for each network. Further, we conduct several profound ML-based performance analyses and apply a naïve transfer approach by synthesizing new networks with well-known graph generators.

IV. Real-World Networks

This section investigates the prediction of the performance metrics solely based on real-world networks.

A. Methodology

In the following, we explain our methodology, i. e., the simulation setup, the dataset, and the utilized ML pipeline.

1) Simulation Setup: The OOS provides an OpenFlow controller implementation comprising a realistic set of functionality, such as topology discovery via the Link Layer Discovery Protocol (LLDP), forwarding mechanisms, and Address Resolution Protocol (ARP) proxy, as well as their respective adaptions for a hierarchical and a flat architecture. In addition to the network traffic caused by LLDP and ARP messages, the main source of traffic are end-devices connected to the switches. The end-devices are based on the *StandardHost* from the INET[7] framework in conjunction with a modified *PingApp*, also from the INET framework. Unless mentioned otherwise, we retain the default parameters of the original OOS for all of the mentioned modules and applications. The exact values of these parameters, the reasoning behind those parameters, and detailed information about the implemented modules can be found in the original OOS paper in [18].

The OOS contains several controller placements for the ported WANs for three different numbers of controllers, namely two, three, and five, placed with respect to C2SL and C2CL with the Pareto Optimal Controller Placement (POCO) framework[8] [33]. As those two latencies are contradicting, this results in a set of Pareto-optimal placements for each network. For the hierarchical architecture the root is always placed with respect to the controller-to-root latency (C2RL).

Each network is simulated for three different numbers of controllers. For each number, the controller placement is altered three times, totaling nine different combinations of placement and number of controllers. For each of these nine simulations, the timeout value of the controller is varied, as different controller implementations may configure different default values on the switches. We vary the timeout for twelve values between $5\,\mathrm{s}$ to $60\,\mathrm{s}$, inspired by default values found in related literature [34]–[36]. Each combination of timeout, placement, and controller amount is simulated four times, each run reflecting a real-world timespan of $10\,\mathrm{min}$. For fifty networks, this results in $3 \cdot 3 \cdot 12 \cdot 4 \cdot 50 = 21{,}600$ simulations for each architecture, or $43{,}200$ runs in total, which reflects the size of our dataset. For one network the OOS has configurations for four numbers of controllers with one placement each, solely optimizing the C2SL. To account for the same amount of data, we perform nine repetitions here.

TABLE I: Coarse overview of the dataset.

	Short Description	Metrics
Dynamic Performance Metrics	• Change throughout a simulation. • Maximum and mean values. • Serve as target variables.	RTT S2C Traffic C2C Traffic
Semi-Dynamic Controller Metrics	• May change for a specific network. • Mainly control plane-related. • Dependent on placement/config. • Serve as input variables.	Timeout # Controllers C2S Latency ...
Static Topology Metrics	• Stay the same for a specific network. • Mainly data plane-related. • Independent from placement/config. • Serve as input variables.	# Switches Betweenness Closeness ...

2) Dataset Description: Table I coarsely summarizes the the extracted features of our simulated datasets. All datasets and an in-depth feature description can be found online[9].

Dynamic performance metrics are the target of our regression, i. e., the values we want to predict. They are highly dynamic, as they change throughout each run, and are measured as a time series. From this time series, the mean and maximum values are contained in the dataset for each of the performance metrics, i. e., round-trip time (RTT), switch-to-controller (S2C) traffic, and controller-to-controller (C2C) traffic. The RTT is the time that a packet needs to reach its destination and vice-versa. The S2C traffic is the traffic induced on the controllers from the data plane, e. g., due to mismatched packets. Lastly, the C2C traffic depicts the required synchronization traffic between all controller instances. For the hierarchical architecture, we focus on the local instances only.

In contrast, *semi-dynamic controller metrics* are used as input features. They may change between runs for the same network due to different configurations, e. g., the specified flow entry timeout or the controller placements. They are mainly control plane-related and thus, among others, include metrics such as C2SL, C2CL, or the sheer number of controllers. All of these metrics are known beforehand, as they do not change throughout the simulation.

Similarly, *static topology metrics* are also known in advance and serve as input. However, they are independent from the actual configuration, as they are merely concerned with the layout of the data plane, i. e., the switches. These metrics can either be *node-based* or *graph-based*. Common node-

[7] https://inet.omnetpp.org/

[8] https://github.com/lsinfo3/poco

[9] https://github.com/lsinfo3/noms2022-sdn-performance-prediction

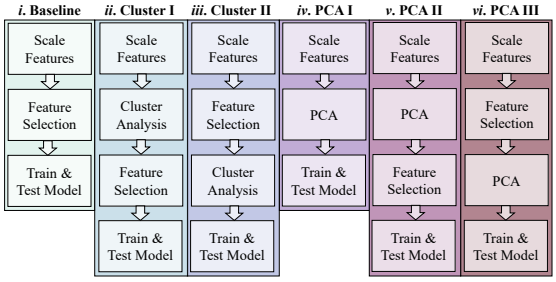

Fig. 1: Compared ML pipelines.

based metrics include the betweenness, closeness, or degree of nodes, whereas exemplary graph-based metrics are diameter or radius, implemented after [37]. As node-based metrics are computed for each switch in the network, we calculate statistical measurements, such as mean, minimum, maximum, standard deviation, and more. Graph-based metrics already concern the whole network and thus consist only of a single value each. If applicable, we calculate these graph metrics as unweighted and weighted versions, i. e., with respect to the number of intermediary devices and with respect to latency.

3) ML Pipeline: To process the extracted data, we define various pipelines. The source code with all hyperparameters and other implementation details can be found online[10].

Training, Test, and Validation Set Splits. As we run four repetitions for each combination of parameters, the splitting of the dataset is crucial, as the performance of the repetitions is similar. Thus, we need to ensure that all four repetitions are either exclusively used for testing **or** training, as otherwise the performance prediction would be trivial. Therefore, we employ the *GroupKFold()* mechanism of the scikit-learn[11] framework [38] to ensure that a *group* only appears in either of the sets, not in both. Here, the group is the network. We apply this splitting technique in a nested cross-validation (CV), with a 5-fold outer and a 5-fold inner CV, used for robustness analysis and hyperparameter optimization, respectively.

Feature Selection and Dimensionality Reduction. As not all features are useful to predict the performance of a specific target, we apply feature selection with *SelectPercentile()* from scikit-learn. Since some of the graph metrics are highly correlated [37], i. e., due to the presence of multicollinearity, we apply dimensionality reduction techniques to reduce the model complexity. Figure 1 depicts the different processing pipelines we apply. The first pipeline defines the baseline, which only applies feature selection. Secondly and thirdly, we combine this with hierarchical clustering, which groups correlated features together and chooses one out of each group. Instead of a cluster analysis and feature selection, the fourth pipeline only applies a Principal Component Analysis (PCA) to transform the feature space into uncorrelated Principal Components (PCs). Lastly, the fifth and sixth pipeline also utilize PCA, once again in conjunction with the feature selection.

ML Models. As a base model we choose a Decision Tree (DT), as it captures non-linear relationships, while still being easily visualizable and explainable. On this basis, we also utilize bagging and boosting techniques, i. e., Random Forest (RF) and AdaBoost (AB), possibly boosting the performance at the cost of more complex models. We leave the tree-based parameters, e. g., tree depth, at their respective default values from scikit-learn. The parameters we optimize are the features of *SelectPercentile()* for all pipelines, except the fourth, which optimizes the amount of PCs instead of features. That is, the fourth pipeline chooses the PCs maintaining the highest variance of the data, instead of choosing features with respect to the target. For pipelines five and six, all PCs are retained and the PCA is merely a measure to remove correlation. The search space of the number of features or PCs considers $10\,\%, 20\,\%, \ldots, 100\,\%$ of the most important features or the retained variance, respectively.

[10]https://github.com/lsinfo3/noms2022-sdn-performance-prediction

[11]https://scikit-learn.org/

B. Evaluation

Table II shows the evaluation results for the mean and maximum prediction for the three performance metrics by depicting the best performing models and pipelines with respect to the Mean Absolute Error (MAE). To quantify the absolute performance, we choose the MAE and Root Mean Squared Error (RMSE). To put the performance into relation with the actual value range, we also depict the Mean Absolute Percentage Error (MAPE) and normalized MAE (nMAE), normalized with the sample mean. We compute the Pearson Correlation Coefficient (PCC) and Spearman Rank Correlation Coefficient (SRCC), to quantify the directional prediction performance, as it may also be of interest if a network performs better than another network.

For all three performance metrics, the prediction of the mean generally performs better than predicting the maximum. The maximum is more prone to outliers, since a single extreme value greatly influences this statistic. Further, the maximum is numerically larger than the mean and thus the error magnitude may also be influenced. In none of the scenarios a PCA-based pipeline is performing best, possibly caused by the nature of the transformation, which only retains the overall variance of the features, not necessarily retaining the importance towards the prediction target. The cluster-based pipelines, however, show promising results. The correlations for all models exceed $90\,\%$ with exception of the maximum RTT, showing feasible results for the prediction of the directional movement.

For the *S2C prediction* we observe similar values for the raw errors for both architectures, as well as for the relative errors, since the S2C traffic is influenced by similar factors. The prediction works well, with MAPEs not exceeding $9.6\,\%$. For both architectures, the most important features chosen by the feature selection in the baseline approach for the mean value prediction concern the switch, e. g., the average number of switches connected to a controller. Each new switch directly introduces a new traffic source, as each switch has a number of clients attached to it. The number of controllers is also identified as important, as they distribute the network traffic. The timeout value is of importance, as each mismatched packet is relayed to the control plane, so maintaining flow

TABLE II: Best performing models and pipelines (according to MAE).

		Hierarchical Architecture								Flat Architecture							
	Metric	Model	Pipeline	MAE[a]	RMSE[a]	MAPE	nMAE	PCC	SRCC	Model	Pipeline	MAE[a]	RMSE[a]	MAPE	nMAE	PCC	SRCC
S2C	Mean	AB	Cluster I	48.45	79.826	0.062	0.065	0.980	0.985	AB	Cluster II	46.63	76.861	0.058	0.063	0.982	0.987
	Max.	AB	Cluster II	716.5	1295.9	0.096	0.114	0.941	0.966	AB	Cluster I	709.6	1320.4	0.093	0.112	0.938	0.967
C2C	Mean	RF	Baseline	2.592	4.3226	0.127	0.097	0.989	0.986	AB	Cluster II	3.633	5.4622	0.032	0.035	0.983	0.986
	Max.	RF	Baseline	8.776	13.189	0.130	0.118	0.975	0.967	RF	Cluster II	13.55	18.644	0.065	0.067	0.917	0.912
RTT	Mean	AB	Baseline	8.174	15.023	0.263	0.206	0.929	0.932	AB	Baseline	5.934	10.455	0.213	0.186	0.948	0.942
	Max.	RF	Cluster II	110.9	224.29	0.472	0.351	0.811	0.872	RF	Baseline	68.58	123.08	0.405	0.280	0.872	0.867

[a] in $\mathrm{B/s}$ for S2C and C2C, and in ms for RTT

entries longer reduces the overall controller load. Several betweenness-related metrics are selected, as a switch of high betweenness is contained in many shortest paths, increasing the refreshing rate of flow entries and ultimately reducing the controller workload. For the maximum value prediction, the maximum number of switches is now also of importance. The number of links is of interest, due to topology discovery messages. For similar reasons, degree-related metrics and node- and edge-disjoint paths are now chosen as well. For the *C2C prediction*, the absolute error is smaller compared to the S2C traffic, as the synchronization traffic is only a fraction of the whole network load, i. e., numerically smaller. The differences of errors for the two architectures are more obvious here, as the synchronization mechanisms differ greatly. The absolute errors are smaller for the hierarchical approach, however, the relative errors are higher. The flat architecture always synchronizes the total network view, so the traffic is numerically larger. However, it is easier to predict, as the hierarchical architecture is more complex. The C2C prediction works well, with nMAEs not exceeding $11.8\,\%$.

For the mean value prediction for the hierarchical approach, the most influential factor is the timeout, as each timed out packet may result in a request to the root. This is followed by several metrics that capture the balance of the S2C mapping, as the hierarchical approach is dependent on how sufficient the local views are. Closeness-, farness- and betweenness-related features are important, as they directly correlate to the probability that flow entries are refreshed as explained in the previous section. Longer paths also increase the probability to cross controller boundaries, resulting in root requests. The latter gains more importance for the maximum, as these topological metrics now rank higher than the balance-related metrics. For the flat approach, the ranking puts less emphasis on the S2C mapping balance, as the placements do not have as much impact, since a global view is maintained anyway.

Lastly, while the *RTT prediction* seems partially feasible, it shows a high degree of heteroscedasticity, i. e., the spread of errors increases drastically with the actual value. This effect may appear since the RTT has a tremendous value range, as network locations range from small countries to networks located over several continents. Naturally, for the RTT prediction the feature selection mainly selects features relating to various latencies, as the RTT is the sum of all occuring delays during transmission. To counteract the lack diverse physical distances, more data is needed, discussed next.

V. Synthetic Networks

Now that we established a baseline, we still need data about the networks, even though the simulative approach circumvents the problem of limited access. To reduce the need of collecting real data, this section examines the performance prediction of real-life networks based on synthetic networks.

A. Methodology

As we employ the same controller applications as before, we focus on the methodology behind the artificial network synthetization and changes to the ML pipeline.

1) Network Generation: We utilize the *NetworkX*[12] [39] package for Python, featuring several algorithms for random graph generation, of which we choose four. Each algorithm takes a number n of nodes as input, which we choose according to a uniform random distribution between 25 and 40 to generate networks of similar sizes to the real networks.

The first and simplest algorithm employed is the Erdős-Rényi model [40], which takes a probability p_{ER} as input, reflecting the chance that an edge is created between two nodes. As WANs typically do not contain too many redundant links due to high geographical distances, we set $p_{\mathrm{ER}} = 5\%$. The second algorithm is the Barabasi-Albert model [41]. Starting with a star graph of $m + 1$, more nodes are added with a preferential attachment to m other high degree nodes. To keep the complexity of the networks low, we set $m = 1$.

Thirdly and fourthly, we employ the Watts-Strogatz [42] and Newman-Watts-Strogatz [43] models. Both start with a ring constellation of nodes, which then are connected to their k or $k - 1$ nearest neighbors, depending if k is even or odd. The original Watts-Strogatz model replaces these edges with a probability p_{WS} with another edge, while the Newman-Watts-Strogatz model merely adds another edge with probability p_{NWS}. For both, we set $k = 2$, as we are not interested in creating highly inter-meshed networks. We set $p_{\mathrm{WS}} = 80\%$ to allow the graph to break the ring structure, while we set $p_{\mathrm{NWS}} = 20\%$, as the ring structure is omitted by adding the random edges.

We generate distances with a uniform and a geometric distribution, each with three different settings, namely *low*, *medium*, and *high* ranges. For each combination of algorithm, distance distribution, and range, we create three random networks, meaning 72 networks in total. We transform the raw

[12] https://github.com/networkx/networkx

TABLE III: Results of training an RF on real versus synthetic data.

		Hierarchical Architecture								Flat Architecture							
		MAE[a]		RMSE[a]		MAPE		nMAE		MAE[a]		RMSE[a]		MAPE		nMAE	
	Metric	Synth.	Real	Synth.	Real	Synth.	Real	Synth.	Real	Synth.	Real	Synth.	Real	Synth.	Real	Synth.	Real
S2C	Mean	45.37	59.08	70.245	86.642	0.058	0.079	0.060	0.081	46.71	63.23	72.878	94.277	0.059	0.083	0.062	0.087
	Max.	874.3	787.4	2180.3	1410.9	0.108	0.108	0.139	0.124	869.7	764.3	2177.1	1385.5	0.106	0.102	0.138	0.121
C2C	Mean	3.645	2.592	6.3390	4.3226	0.168	0.127	0.136	0.097	5.257	4.160	8.6172	6.0428	0.047	0.037	0.050	0.040
	Max.	9.856	8.776	14.894	13.189	0.150	0.130	0.132	0.118	17.88	14.82	30.601	21.471	0.081	0.071	0.090	0.074
RTT	Mean	10.92	9.396	19.178	16.269	0.413	0.318	0.276	0.236	7.479	6.683	13.537	11.785	0.211	0.237	0.227	0.204
	Max.	108.5	122.5	249.67	247.66	0.394	0.479	0.343	0.380	88.04	68.58	153.77	123.08	0.386	0.405	0.366	0.280

[a] in $\mathrm{B/s}$ for S2C and C2C, and in ms for RTT

graphs into the corresponding OMNeT++/C++ representation, parameterizing the number of controllers with the same three values as for the real-world networks, i. e., two, three, and five. For each of the 72 networks and each amount of controllers, we generate three different controller placements, with the placement being chosen at random. The switches are assigned to the closest controller with respect to the geographical distance. We vary the timeout for twelve different values and each combination of timeout, placement, and controller amount is repeated four times, resulting in $3 \cdot 72 \cdot 3 \cdot 12 \cdot 4 = 31{,}104$ simulation runs for each architecture, or $62{,}208$ runs in total.

2) ML Pipeline: As we now have a predefined split between train and test set, i. e., random and real networks, we do not employ an outer CV, only an inner CV. Lastly, we apply the baseline pipeline with an RF to not lose any information.

B. Evaluation

Table III shows the results of training the RF with synthetic data to predict the performance of the real-world networks. We compare this to an RF trained with the baseline pipeline and real data for comparability's sake. The respective better performance is highlighted in the table.

For both architectures, the absolute errors for the *S2C prediction* for the mean value show promising results, also manifesting in slightly lower relative errors. For the maximum value, we see an increase of the absolute errors. While not quite achieving the results of the baseline, this may still be feasible, as the relative errors approximate the baseline. For the *C2C prediction*, the MAEs for both architectures grow overall, also manifesting in the relative errors increasing by a few percentage points. However, similar to the S2C traffic, the results still roughly approximate the training on real-world networks and thus are still useful. Lastly, the *RTT prediction* illustrates mixed results for this naïve transfer approach. For the maximum value for the hierarchical architecture this approach outperforms the baseline in all aspects besides the RMSE, while the mean value prediction underperforms. For the flat architecture, only both MAPEs decrease, while the rest of the errors increase.

To analyze the weaknesses of the synthesized networks, we merge the datasets, label them either *real* or *synthetic*, drop the performance metrics, and train an RF to classify a network accordingly. The results show that an RF can distinguish between real and synthetic networks with $89.8\,\%$ accuracy. The synthetic networks generally contain longer paths, which influences many topological metrics, as they are mostly distance-based. Furthermore, the chosen distribution of the distances does not reflect the real-world networks, as WANs may contain many short and few extraordinarily long links, e. g., for inter-continental connections.

While this first approach is not optimized yet, we are still able to approximate the baseline approach. This enables not only the performance prediction during network planning, but also without the need to collect real-world network data. The generated networks do not capture the characteristics of the real-world networks fully. Thus, we hope to generate more realistic network graphs in the future to improve the prediction.

VI. Conclusion

In this work, we evaluated the feasibility of predicting the performance of SDN-enabled networks based on network properties, which are known during the design phase. For this, we simulated over $100{,}000$ runs, which sum up to a real-world time of over two years. As a first scenario, we evaluated the performance prediction based on a set of real-world WANs for two different distributed SDN architectures, six different ML preprocessing pipelines, and three different performance metrics. We showed that it is feasible to predict the performance by the extracted features. As a second scenario, we created networks with well-known graph generators to reduce the need of real-world data acquisition. We proceeded to train the ML models on this synthetic data and evaluated the three defined performance metrics again. While the synthetic networks already showed promising results, there is still room for improvement, as the generated networks did not capture all characteristics of real-world WANs.

In the future, we aim to vary more factors, e. g., more realistic traffic patterns and workloads of varying magnitude. A more thorough investigation of the structure of real-world WANs can increase the quality of the synthesized data. We will investigate more sophisticated graph generation methods, such as [44] or Generative Adversarial Networks (GANs) [45].

Acknowledgment

This work has been funded by the German Federal Ministry of Education and Research (BMBF) as part of the project WINTERMUTE (16KIS1129).

REFERENCES

[1] M. He, M.-Y. Huang, and W. Kellerer, "Optimizing the flexibility of SDN control plane," in *2020 IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2020, pp. 1–9.

[2] E. Kaljic, A. Maric, P. Njemcevic, and M. Hadzialic, "A survey on data plane flexibility and programmability in software-defined networking," *IEEE Access*, vol. 7, pp. 47 804–47 840, 2019.

[3] K. Poularakis, Q. Qin, L. Ma, S. Kompella, K. K. Leung, and L. Tassiulas, "Learning the optimal synchronization rates in distributed SDN control architectures," in *2019 IEEE International Conference on Computer Communications (InfoCom)*. IEEE, 2019, pp. 1099–1107.

[4] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 42, no. 4, pp. 473–478, 2012.

[5] M. Darianian, C. Williamson, and I. Haque, "Experimental evaluation of two OpenFlow controllers," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE, 2017, pp. 1–6.

[6] L. Zhu, M. M. Karim, K. Sharif, C. Xu, F. Li, X. Du, and M. Guizani, "SDN controllers: A comprehensive analysis and performance evaluation study," *ACM Computing Surveys (CSUR)*, vol. 53, no. 6, pp. 1–40, 2020.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 38, no. 2, pp. 69–74, 2008.

[8] G. Zhao, L. Huang, Z. Yu, H. Xu, and P. Wang, "On the effect of flow table size and controller capacity on SDN network throughput," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–6.

[9] C. Metter, M. Seufert, F. Wamser, T. Zinner, and P. Tran-Gia, "Analytical model for SDN signaling traffic and flow table occupancy and its application for various types of traffic," *2017 IEEE Transactions on Network and Service Management (TNSM)*, vol. 14, no. 3, pp. 603–615, 2017.

[10] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking (INM/WREN)*, vol. 3, 2010.

[11] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012, pp. 19–24.

[12] A. Kumari and A. S. Sairam, "A survey of controller placement problem in software defined networks," *arXiv preprint arXiv:1905.04649*, 2019.

[13] R. Barrett, A. Facey, W. Nxumalo, J. Rogers, P. Vatcher, and M. St-Hilaire, "Dynamic traffic diversion in SDN: testbed vs Mininet," in *2017 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2017, pp. 167–171.

[14] J. Alcorn, S. Melton, and C. E. Chow, "SDN On-The-Go (OTG) physical testbed," in *2017 IEEE Conference on Dependable and Secure Computing (DSC)*. IEEE, 2017, pp. 202–208.

[15] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE Journal on Selected Areas in Communications (J-SAC)*, vol. 29, no. 9, pp. 1765–1775, 2011.

[16] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 12)*, 2012.

[17] S. Herrnleben, P. Rygielski, J. Grohmann, S. Eismann, T. Hoßfeld, and S. Kounev, "Model-based performance predictions for SDN-based networks: A case study," in *International Conference on Measurement, Modelling and Evaluation of Computing Systems (MMB)*. Springer, Cham, 2020, pp. 82–98.

[18] N. Gray, T. Zinner, S. Gebert, and P. Tran-Gia, "Simulation framework for distributed SDN-controller architectures in OMNeT++," in *International Conference on Mobile Networks and Management (MONAMI)*. Springer, 2016, pp. 3–18.

[19] S.-Y. Wang, H.-W. Chiu, and C.-L. Chou, "Comparisons of SDN OpenFlow controllers over EstiNet: Ryu vs. NOX," in *The Fourteenth International Conference on Networks (ICN)*, vol. 256, 2015.

[20] R. Jawaharan, P. M. Mohan, T. Das, and M. Gurusamy, "Empirical evaluation of SDN controllers using Mininet/Wireshark and comparison with Cbench," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2018, pp. 1–2.

[21] A. Van Bemten, N. Dheric, A. Varasteh, A. Blenk, S. Schmid, and W. Kellerer, "Empirical predictability study of SDN switches," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2019, pp. 1–13.

[22] P. Rygielski, M. Seliuchenko, and S. Kounev, "Modeling and prediction of software-defined networks performance using queueing Petri nets," in *Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques (SIMUTOOLS)*, 2016, pp. 66–75.

[23] K. Sood, S. Yu, and Y. Xiang, "Performance analysis of software-defined network switch using $M/Geo/1$ model," *IEEE Communications Letters (COMML)*, vol. 20, no. 12, pp. 2522–2525, 2016.

[24] J. Ansell, W. K. Seah, B. Ng, and S. Marshall, "Making queueing theory more palatable to SDN/OpenFlow-based network practitioners," in *2016 IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2016, pp. 1119–1124.

[25] N. Gray, K. Dietz, M. Seufert, and T. Hossfeld, "High performance network metadata extraction using P4 for ML-based intrusion detection systems," in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2021, pp. 1–7.

[26] C. Hardegen and S. Rieger, "Prediction-based flow routing in programmable networks with P4," in *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 2020, pp. 1–5.

[27] A. Filali, Z. Mlika, S. Cherkaoui, and A. Kobbane, "Preemptive SDN load balancing with machine learning for delay sensitive applications," *IEEE Transactions on Vehicular Technology (TVT)*, vol. 69, no. 12, pp. 15 947–15 963, 2020.

[28] R. Pasquini and R. Stadler, "Learning end-to-end application QoS from OpenFlow switch statistics," in *2017 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2017, pp. 1–9.

[29] T. Abar, A. B. Letaifa, and S. E. Asmi, "Quality of Experience prediction model for video streaming in SDN networks," *International Journal of Wireless and Mobile Computing (IJWMN)*, vol. 18, no. 1, pp. 59–70, 2020.

[30] B. Kheibari and M. Sayıt, "Quality estimation for DASH clients by using deep recurrent neural networks," in *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 2020, pp. 1–8.

[31] S. Jain, M. Khandelwal, A. Katkar, and J. Nygate, "Applying big data technologies to manage QoS in an SDN," in *2016 12th International Conference on Network and Service Management (CNSM)*. IEEE, 2016, pp. 302–306.

[32] N. Gray, K. Dietz, and T. Hossfeld, "Simulative evaluation of KPIs in SDN for topology classification and performance prediction models," in *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 2020, pp. 1–9.

[33] D. Hock, S. Gebert, M. Hartmann, T. Zinner, and P. Tran-Gia, "POCO-framework for Pareto-optimal resilient controller placement in SDN-based core networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–2.

[34] A. Azzouni, O. Braham, T. M. T. Nguyen, G. Pujolle, and R. Boutaba, "Fingerprinting OpenFlow controllers: The first step to attack an SDN control plane," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–6.

[35] A. Zarek, Y. Ganjali, and D. Lie, "OpenFlow timeouts demystified," *Univ. of Toronto, Toronto, Ontario, Canada*, 2012.

[36] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011, pp. 254–265.

[37] J. M. Hernández and P. Van Mieghem, "Classification of graph metrics," *Delft University of Technology: Mekelweg, The Netherlands*, pp. 1–20, 2011.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[39] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using NetworkX," Los Alamos National Lab (LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[40] P. Erdös and A. Rényi, "On the evolution of random graphs," in *The Structure and Dynamics of Networks*. Princeton University Press, 2011, pp. 38–82.

[41] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[42] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[43] M. E. Newman and D. J. Watts, "Renormalization group analysis of the small-world network model," *Physics Letters A*, vol. 263, no. 4-6, pp. 341–346, 1999.

[44] M. Seufert, S. Lange, and T. Hoßfeld, "More than topology: Joint topology and attribute sampling and generation of social network graphs," *Computer Communications*, vol. 73, pp. 176–187, 2016.

[45] W. Liu, P.-Y. Chen, F. Yu, T. Suzumura, and G. Hu, "Learning graph topological features via GAN," *IEEE Access*, vol. 7, pp. 21 834–21 843, 2019.