# Simulative Evaluation of KPIs in SDN for Topology Classification and Performance Prediction Models

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

***Abstract*—In recent years Software-defined Networking (SDN) has gained increased popularity by reducing the complexity of management operations and increasing the performance of networks. As a result, the number of purchasable SDN devices and their deployment in real world network constantly rises, making a thorough understanding of their behaviors and interactions even more important. In this work we analyze distributed controller architectures via simulation, identify Key Performance Indicators (KPIs), classify various network topologies with respect to their impact on SDN, as well as create a prediction model to estimate the overall performance of the overall SDN ecosystem, which can be used for network planning.**

***Index Terms*—Software-defined Networking (SDN), Event-based Simulation, OMNeT++, OpenFlow, Performance Prediction, Network Optimization, OpenFlow OMNeT Suite, Linear Regression, Principal Component Analysis (PCA)**

## I. Introduction

Software-defined Networking (SDN) has been proven to increase the scalability and flexibility not only in data centers, but is also applied to Wide Area Networks (WANs) as seen in Google's B4 backbone [1]. Due to its centralized network configuration implemented by separating the control and data plane, SDN-enabled networks may be managed at a central point, allowing for quick adaptations via dedicated APIs.

The OpenFlow protocol is often regarded as one of the pioneers to standardize the communication between the data and control plane in software-defined networks. Here, the OpenFlow controller is the decision maker, i.e., represents the control plane, while OpenFlow switches are the executive authorities and thus, constitute the data plane. Nowadays, the controller is usually distributed among several physical instances to avoid performance bottlenecks [2] and to further enhance the resiliency of the network, while still maintaining a logical centralized view.

Yet, these distributed environments impose additional challenges regarding their performance, their organizational architecture, the required synchronization overhead as well as the placement of the individual controller instances. On their own, these challenges have been investigated by several papers. Yet a thorough investigation of their impact on the entire SDN ecosystem in combination with varying controller applications, data plane traffic patterns, network topologies as well as the applied distributed controller architecture remains outstanding, which is vital for efficient network planning.

The contribution of this work is (1) a thorough investigation of distributed architectures, (2) the identification of Key Performance Indicators, (3) a classification of network topologies and finally (4) the creation of a performance prediction model, based on the prior findings.

To achieve these goals and to evaluate a myriad of network configurations without the potentially high costs and limitations of real world test beds, we utilize the OpenFlow OMNeT++ Suite [3] as simulation framework. The suite has been extended within this work by a library of network topologies as well as more sophisticated modules for traffic generation and graph analysis. The KPIs are extracted from the obtained simulation results and analyzed by using correlation as well as Principal Component Analysis (PCA), which enable us to create a performance prediction model via linear regression.

The remainder of this work is structured as follows. In Section II we start by coarsely outlining the OpenFlow specifications, distributed controller architectures, as well as the core functionalities of the OpenFlow OMNeT++ Suite. This is continued by summarizing related work in Section III. Section IV provides a description of the improvements to the OpenFlow OMNeT++ Suite, followed by details about the utilized scenarios and configurations of the simulation, before presenting our evaluation results in Section V. This covers the identification of the KPIs, the classification of various networks, as well as the derived model for predicting the network's performance. Finally, the paper is summarized in Section VI and we provide an outlook for future research.

## II. Background

In this section we provide the required background of this work, by briefly detailing the OpenFlow protocol and distributed controller architectures as well as their implementation within in the OMNeT++ simulation framework.

### A. OpenFlow & Controller Architectures

Being one of the first enablers of SDN, the OpenFlow protocol[1] specifies the communication between the control and data plane, i.e., it acts as a southbound interface connecting

978-3-903176-31-7 

[1] https://www.opennetworking.org/software-defined-standards/

the OpenFlow controller and the OpenFlow switches. The latter are equipped with so-called Flow tables against which incoming packets are matched. If an entry is successfully matched, the packets are subject to its configured actions, which determine its further course and is known as the fast path. If no matching entry is found, the essential information of the mismatched packet is extracted and relayed to the controller, which will then decide how to handle the packet and may set additional configurations within the reporting switch. As this process induces additional delay, it is referred to as the slow path. Therefore, it is highly beneficial to process as many packets as possible within the fast path to minimize the delay and to reduce the overall load imposed on the controller. In addition to the data plane traffic and applied forwarding mechanisms, further traffic may be afflicted to the control plane depending on the configured control plane applications such as topology discovery or network monitoring. To mitigate a possible overloading of the OpenFlow controller, distributed architectures aim to divide the traffic among several instances. Existing architectures can be divided into two groups: horizontal and hierarchical approaches [3]. Whereas the horizontal approach requires constant synchronization and hence is able to keep all instances on par with each other, the hierarchical approach divides the authority among all instances grouped by hierarchy level, i.e., the root instance is the only node with the full network view. Representatives for the horizontal and hierarchical approach are HyperFlow [4] and OpenDaylight[2] as well as Kandoo [5] and ONOS[3] respectively.

### B. Event-based Simulation & OpenFlow OMNeT++ Suite

Having limited access to large scale SDN-enabled testbeds capable of establishing various topologies, we utilize the merits of event-based simulation in this work to overcome this restraint. For this we incorporate the OMNeT++[4] framework, which is component-based, easy to extend, and is mainly focused on the simulation of networks. In addition, we utilize the INET[5] and OpenFlow OMNeT++ Suite (OOS) [3] extensions, which implement common network protocols and the core functionality of OpenFlow respectively. The latter is also equipped with modules for HyperFlow and Kandoo as well as a collection of control and data plane applications. Yet, to achieve the goals of this work, OOS has been further extended, which is detailed in Section IV.

## III. Related Work

In this section we discuss related research by stating similarities and differences to this work.

*OpenFlow Performance Evaluation:* The performance of SDN has been subject of research papers from the very start. Yet, most publications focus only on a selective part of the SDN ecosystem, which can be summarized in analogy to the SDN architecture, i.e., the data and control plane. The authors of [6]–[9] investigate how to improve the overall Quality of Service (QoS) and Quality of Experience (QoE) of the network by utilizing the advantages of the enhanced control mechanisms of SDN in combination with expert knowledge of the applications. [10], [11] zero in on the processing times of SDN-enabled switches for various tasks, revealing a wide heterogeneity among switching operations and hardware vendors. In contrast, the papers [12]–[15] focus solely on the SDN controller, detailing a large variety regarding the performance of the particular implementations. While adopting certain test methodologies and metrics, the focus of this work is not a single component but rather the entire SDN ecosystem and the interconnection of all influence factors.

*OpenFlow Performance Optimization:* The SDN controller performance is further enhanced via code optimization and parallelization as stated in [16]–[18]. Another approach is by shifting static and commonly used rules directly to the switch [19], [20] or by incorporating several distributed controller instances, which share the load and maintain a logically centralized view [4], [5]. In this paper, we examine distributed controller architectures and show their impact on the overall network performance, as this has not been investigated exhaustively with regards to different network topologies and traffic patterns. Furthermore, we reckon that utilizing the fast path can significantly reduce the load on the SDN controllers and reduce the end-to-end latency within the network.

*Controller Placement:* The performance and resiliency of the network is further influenced by the placement of the SDN controllers known as as the Controller Placement Problem (CPP). This is investigated in [21]–[23] by using heuristics or exhaustive search to optimize a selective set of metrics such as the controller-to-switch latency (C2SL). For our simulation, we incorporate optimized placements with respect to the C2SL and in addition, the results of this work demonstrate that simple metrics from which the placements are derived may be insufficient as they are unable to model vital interactions and side effects, leading to sub par performance results.

*Network Modeling:* Former analysis have utilized more abstract network models, which often have their origin in graph theory. Studies have shown that the graph's topology influences the resiliency and performance [24], [25], which is reflected by various graph metrics. A selection of graph metrics is summarized in [26], which we also implement within this work and investigate which of these are incisive for SDN-enabled networks.

The ability to predict the performance of a network is valuable, as it allows for cost effective network design and extends the reaction times in dynamic systems to critical events. The authors of [27] group prediction methods for legacy networks by system identification, time series analysis, machine learning as well as queuing theory. The latter is applied by [28], [29] to the controller/switch interaction of SDN. In contrast, we use linear regression based on the results of a full simulation for predicting the performance of the entire SDN ecosystem with respect to the network topology, distributed controller architecture and traffic pattern.

[2]https://www.opendaylight.org/
[3]https://www.opennetworking.org/onos/
[4]https://omnetpp.org/
[5]https://inet.omnetpp.org/

## IV. Simulation Description and Setup

In the following we detail our improvements to the OpenFlow Suite and describe our simulation setup.

*Extension of the OpenFlow Suite:* To investigate our research questions, we had to extend the OpenFlow OMNeT++ Suite (OOS), which is briefly discussed in the following. At first the computation of the graph metrics has been implemented within the *GraphAnalyzer* module, which is triggered at the beginning of every simulation and respects the separation of the data and control plane of SDN scenarios. In total the module derives 20 distinct metrics and records the results as scalars. We ported 50 networks based on their size from the internet Topology Zoo[6] (ITZ) to OOS, in order to investigate the influence of topological features in addition to the already featured networks AL2S and FatTree, which are extreme opposites. Whereas the Fat-tree is a common topology within data center environments due to its symmetrical and fault tolerant design, we selected topologies from the ITZ, which represent WANs and have at least 25 up to 40 switch nodes to increase the variety of our investigations. Another influence factor is the underlying traffic pattern generated by the individual hosts in the network. Here, OOS already features a random ping application and a TCP traffic generator, which both choose the source and destination of every flow by an uniform random distribution. As this may have a negative impact on the distributed controller architectures, we extended the traffic generators to respect the controller-to-switch affiliation by introducing locality-aware traffic patterns. The above changes and other slight improvements have been merged with the OOS core and are publicly available[7].

*Simulation Topology and Setup:* In the following, the underlying topologies, the module configuration, and the general simulation setup, as used in this work, is described.

For all simulated topologies, we computed the appropriate placements with respect to an average minimal C2SL for the varying amounts of controllers and the investigated distributed architectures, i.e., HyperFlow and Kandoo. For the networks taken from the ITZ, the Kandoo root is placed according to the minimal distance to the local instances. In case of the Fat-tree scenario, the controllers are all connected to a regular Ethernet switch, which also connects each switch in the data plane enabling a more balanced controller-to-switch distribution. In total 1712 simulation runs have been conducted.

The controller module is governed by several applications with their corresponding alterations for the distributed architectures, as depicted in Figure 1a and 1b. This consists of the respective distributed controller agents which are responsible for managing the synchronization. Furthermore, the *LLDPAgent* performs network topology discovery and the *LLDPForwarding* as well as the *LLDPMinHop* applications are responsible for the packet forwarding mechanism by using the discovered information. Last, the *ARPResponder* maintains an IP-to-MAC address mapping and functions as ARP proxy. The stated applications have been chosen, as they either perform a vital function within the network or are commonly deployed by default in real-world SDN controllers to reflect a realistic control plane application mix.

Each simulation run is configured to reflect 30 minutes of real-world measurements and is repeated four times. The remaining parameters are set to their default values as specified by the OOS and have been additionally summarized[8].

Whereas, the graph metrics are collected at the initialization stage, the performance metrics are measured during the simulation. In this work, we focus on four performance metrics to represent the key actors of a SDN-enabled network. The amount of synchronization traffic measures the performance of the distributed controller architecture. In addition, the controller and OpenFlow switch efficiency are in coherence with the amount of induced control plane traffic and the Flow table hit-to-miss ratio, respectively. Last, the round-trip time and the connection setup time capture the performance of the data plane applications as they reflect the end-to-end latency of the entire system from the user's perspective.

## V. Evaluation

In this section we evaluate the results obtained in the simulation runs to determine the main influence factors on the network's performance, which are then used to classify various network topologies to finally create a prediction model for the network performance based on topological features.

### A. Identification of Key Performance Indicators (KPIs)

We start by presenting the results of a parameter study based on the simulation runs to derive KPIs. For this we vary the number of controllers, the distributed controller architecture, the applied traffic patterns as well as the network topology. In this section we focus on the Fat-tree and Advanced Layer 2 Services (AL2S) topologies only, as these reflect opposite design patterns and the AL2S is popular in research papers, which allows for further comparison. A more detailed examination of the network topologies is presented in the next section.

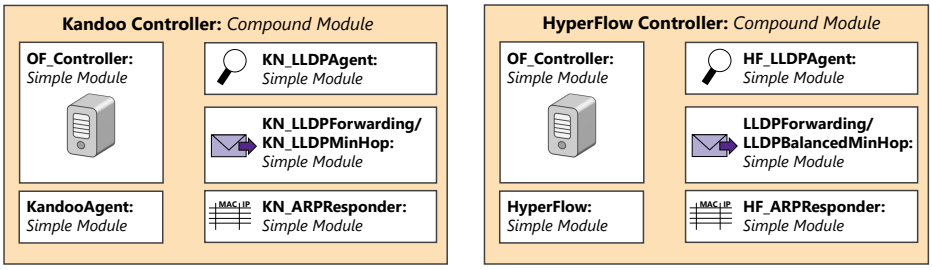
The results are shown in Table I and display the mean values over 4 simulation runs. All results have been verified for statistical significance by computing the 95% confidence intervals. All confidence intervals are small, due to the length of the simulated time span and thus have been omitted for the sake of brevity. In addition, we dismiss the warm up phase, which consists of the first 250 s, after which a steady state is reached within the simulation as all switches have connected to their respective controllers and the first rounds of topology discovery have been completed.

*1) Fat-Tree:* Focusing on the control plane traffic, the table depicts a reduced load at the specific controller instances for a rising number of controllers. This is due to the fact that they now have to manage less switches. However, if one aggregates the traffic over all controllers different trends for Kandoo

[6]http://www.topology-zoo.org/dataset.html

[7]https://github.com/lsinfo3/OpenFlowOMNeTSuite/

[8]https://github.com/lsinfo3/cnsm2020




(a) Kandoo (b) HyperFlow

Fig. 1: Adjusted controller compositions for Kandoo (a) and HyperFlow (b).

TABLE I: Measured performance metrics for Kandoo and HyperFlow with varying number of controllers (2c-5c).

| | | | Fat-Tree | | | | AL2S | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 2C | 3C | 4C | 5C | 2C | 3C | 4C | 5C |
| Kandoo | **Control P. Traffic** | (B/s) | 2639.7 | 1684.0 | 1246.5 | 976.10 | 2095.8 | 1395.7 | 1066.3 | 858.71 |
| | **Agg. Control P. Traffic** | (B/s) | 5279.4 | 5051.9 | 4985.9 | 4880.5 | 4191.6 | 4187.1 | 4265.6 | 4293.5 |
| | **Hit-to-Miss Ratio** | (%) | 73.518 | 74.801 | 78.250 | 75.795 | 83.916 | 84.076 | 83.611 | 83.499 |
| | **Round-Trip Time** | (ms) | 0.6667 | 0.6621 | 0.7415 | 0.6589 | 93.714 | 111.30 | 119.93 | 112.05 |
| | **Sync. Traffic** | (B/s) | 178.62 | 141.72 | 113.88 | 95.117 | 161.51 | 122.91 | 120.92 | 103.08 |
| | **Agg. Sync. Traffic** | (B/s) | 357.24 | 425.17 | 455.52 | 475.59 | 323.02 | 368.74 | 483.68 | 515.41 |
| HyperFlow | **Control P. Traffic** | (B/s) | 3323.8 | 2522.2 | 2139.6 | 1820.1 | 2134.8 | 1477.4 | 1161.3 | 921.14 |
| | **Agg. Control P. Traffic** | (B/s) | 6647.6 | 7566.6 | 8558.4 | 9100.6 | 4269.6 | 4432.2 | 4645.3 | 4605.7 |
| | **Hit-to-Miss Ratio** | (%) | 64.508 | 59.370 | 50.798 | 46.642 | 83.586 | 82.925 | 82.057 | 82.216 |
| | **Round-Trip Time** | (ms) | 0.6942 | 0.7280 | 0.7655 | 0.7852 | 78.183 | 79.948 | 82.368 | 70.483 |
| | **Sync. Traffic** | (B/s) | 157.48 | 169.59 | 181.61 | 193.60 | 139.23 | 151.20 | 162.91 | 175.09 |
| | **Agg. Sync. Traffic** | (B/s) | 314.97 | 508.77 | 726.43 | 968.02 | 278.46 | 453.61 | 651.62 | 875.46 |

and HyperFlow are visible. The total control plane load is increasing for HyperFlow the more instances are utilized, whereas it is decreasing for Kandoo. While the HyperFlow instances all maintain a global network view, this view may be inconsistent throughout all devices for certain time periods. Since the controllers synchronize in cycles, a specific instance may not have yet received the information of another controller and hence may perform sub-optimal decisions. In contrast, in a Kandoo environment the root is the only device with a global view and the sole authoritative controller for global decisions. This varying behavior results in a different granularity and consistency regarding the forwarding configuration, which directly affects the requests to the controller. The less consistent the forwarding decisions are, the less likely it is that flow entries are refreshed and remain active within the flow tables. The decrease of the overall traffic for Kandoo is explained by the fact that if more controllers are utilized, more paths are configured by the root instead of the local instances due to insufficient local views. In return, this results in a more consistent forwarding configuration. Whereas for HyperFlow each individual instance becomes more dependent on the others, since each controller only configures paths under their respective control and relies on the precondition that all controllers have the same knowledge base to determinedly compute the same path configuration.

This effect is also visible when examining the Flow table hit-to-miss ratio. The hit ratio for Kandoo increases minimally for more controllers with the exception of the four controller scenario, while it decreases for HyperFlow. A packet transmission may cause more than one table miss in a HyperFlow environment, since a packet may be forwarded before the complete path is established, due to asynchronous authoritative controller instances, which further worsens the hit ratio.

The anomaly of the hit-to-miss ratio for Kandoo utilizing four controllers is caused by a logical error within the path configuration for one of the four simulation runs. The network traffic is routed over multiple equivalent paths, hence, it may occur that old Flow entries cause a message to cycle through the network longer than needed. Since this ping-pong behavior mostly results in table hits, this does not affect the control plane traffic and is an example for the complex interaction of the individual SDN components.

The round-trip times are directly correlated to the hit ratio. For Kandoo a minor decrease is observed, again, with exception to the discussed four controller scenario, where packets recirculate in the network. For HyperFlow an increase of round-trip times is shown, since the probability for a packet to be stalled, due to multiple table misses increases.

Finally, the synchronization traffic of the two architectures diverges. For Kandoo, the required synchronization decreases at each instance the more controllers are utilized, as the load is balanced over more devices. However, when accumulating the traffic over all controllers, an increase is observed. Since the network is more fragmented, more synchronization with the root is required, because the partial views are insufficient. For HyperFlow, the synchronization traffic increases for more utilized instances on every level, as all controllers maintain a global view and have to report their knowledge to all other instances. Hence, the baseline of the required synchronization cannot be decreased.

*2) AL2S:* Similar to the Fat-tree, the control plane traffic at the individual instances decreases for the AL2S topology as less switches are handled by each controller. However, the table depicts two major differences compared to the Fat-tree scenario. First, the controller traffic for both architectures is significantly lower in the AL2S scenario. This is caused by a more consistent forwarding configuration, which is applied due to the lack of symmetry within the network and hence the abundance of equal costs paths over which the load could be balanced. This causes the controllers to only configure a single path to route the traffic in the network. As a result, this creates less divergent Flow entries and therefore, increases the chance that a specific entry gets refreshed. This provides fewer table misses and in return, less traffic is relayed to the controller.

The second difference compared to the Fat-tree topology is the similar performance of both architectures, hence the Fat-tree topology happens to be beneficial for Kandoo and disadvantageous for HyperFlow. The improved performance of HyperFlow is also a result of the different forwarding mechanism described before, since the load balancing over multiple paths amplified the weaknesses of HyperFlow. Nonetheless, while performing more similar to Kandoo, HyperFlow still induces up to $380\,\mathrm{Bps}$ more aggregated traffic, which is again attributable to the asynchronous authoritative among the HyperFlow instances.

The observed variations induced to the aggregated control plane traffic for both architectures are caused by the asymmetry in the topology, which results in imbalanced controller-to-switch mappings, as we did not optimize for balanced switches. Thus, we see no monotonous trends like for the Fat-tree, where the switches where divided evenly among the controllers.

As before, the hit ratio is in accordance to the aggregated control plane traffic and shows the same non-linear trend, since a table miss results directly in a request to a controller.

Another difference to the Fat-tree is that the round-trip time increases for Kandoo when utilizing more controllers, with the exception of the five controller case. The round-trip time first increases, since a Flow entry time out is less likely to be handled by a local instance in a more fragmented network and thus, the root is asked for further assistance more often. In contrast to the Fat-tree scenario, this has a noticeable influence on the round-trip times due to higher physical distances between the sites. E.g., the average link delay for the shortest paths from local instances to the Kandoo root for two controllers is remarkably lower with a value of around $6.5\,\mathrm{ms}$, while the delay ranges from $8.4\,\mathrm{ms}$ to $9.2\,\mathrm{ms}$ for other scenarios. The improvement of the round-trip time when deploying five controllers is attributable to the fairer switch-to-controller mapping, regarding the amount of switches assigned to an individual controller. For example, in the three controller scenario, two thirds of the switches are assigned to the first controller. This imbalance is further increased by asymmetric traffic flows, e.g., the node representing Chicago handles around $2.7$ times more data plane traffic than the cumulative mean of the remaining switches. Thus, controllers managing more and/or switches handling extensive data plane traffic, have a higher chance of inducing waiting times onto incoming requests, which are ultimately reflected by higher round-trip times. For HyperFlow, the round-trip times are generally lower compared to Kandoo. This is due to the fact that no information is requested from a root controller. However, as seen previously, the more controllers are utilized, the more dependent each controller is on the other instances. Thus, the round-trip times increase minimally up to four controllers. Then, just like for Kandoo, the times decrease for the five controller scenario, due to more balanced switch assignments.

The synchronization traffic behaves similarly as for the Fat-tree topology, i.e., whereas it decreases for Kandoo's local instances, it rises for HyperFlow at the individual instances for an increasing number of controllers. Again, the cumulative synchronization traffic increases for both distributed controller architectures. Yet, the imbalance of switch mappings is also reflected in these results, especially looking at Kandoo, as the synchronization load at a single instance only decreases minimally from three to four controllers. This is further reflected in the accumulated synchronization traffic, since the increase from three to four controllers is remarkably higher than for the other scenarios. Examining the controller placement more closely, the deployment for two and three controllers is similar, as both scenarios feature a controller that maintains two thirds of the traffic. Adding a third controller results in an increase of control plane traffic due to a stronger fragmentation of the switch mappings. Yet, the configuration of four and five instances is again similar, since they share three controller placements, thus resulting in similar balanced traffic patterns.

*3) Data Plane Traffic Patterns:* To examine the influences of varying data plane traffic more closely, we first evaluate the impact of TCP flows instead of pings and then focus on traffic locality properties, i.e., the amount of flows established within or across controller boundaries. For the sake of brevity the results are merely summarized with respect to the main observed differences compared to the previous section.

Instead of the previously used traffic generator which uses deterministic intervals to establish connections, the TCP traffic generator uses a gamma distribution, which has been configured to model the traffic pattern of a real-world dormitory [3]. This is results in connections being around $6.5$ times less frequent while inducing around $50\%$ more traffic into the data plane. For both distributed controller architectures and

topologies similar trends as seen before are observed. The main difference here is that even though more data plane traffic is present, the control plane traffic is reduced to a third in comparison when using the deterministic ping pattern as before. This is due to the fact that the flow table hit ratio is now far higher ranging from $95\%$ to $96\%$ and from $87\%$ to $91\%$ for Kandoo and HyperFlow respectively. In return, this is caused by the nature of TCP flows, which consist of several packets in both directions within a short time frame, hence keeping the table flow entries active once set.

As shown previously, side effects may occur if traffic is routed across controller boundaries. To investigate this in more detail, we adapt the selection of communication partners within the traffic generator, to respect the controller-to-switch mappings in order to define a certain percentage of flows, which are kept local, hence not crossing a controller boundary. For this value a parameter study is performed starting from $100\%$ and ranging down to $0\%$ in steps of $20\%$. As expected the control plane traffic for both architectures and deployed number of controllers first rises with a decreasing locality value, until it hits a turning point from which it starts declining. This behavior results from the fact that now hosts are favored due to their association to a certain controller boundary and hence their respective flow table entries are refreshed with an higher likelihood. This effect is even more emphasized in scenarios using the AL2S topology or a specific number of deployed controllers, which feature more unbalanced controller-to-switch mappings. Therefore, the turning point is different for each topology and number of controllers. For the flow table hit ratio similar observations are made, which also impact the round-trip times.

### *B. Classification of Network Topologies*

As depicted in Table I, the network topology has a large impact on the overall performance of the SDN ecosystem. Hence, we perform an in-depth analysis of selected networks of the Internet Topology Zoo to generalize our results and to form a basis for creating a prediction model. We first analyze the correlation of the performance metrics with the graph metrics, followed by a visualization and classification of the individual networks via Principal Component Analysis (PCA).

Note that for all analyses the control plane and synchronization traffic have been normalized with the number of hosts in the network. This reduces the effect of the sheer number of participants on the performance and augments the impact of the underlying topology. Henceforth, the terms control plane traffic and synchronization traffic refer to the normalized performance per host.

*1) Correlation Analysis:* Figure 2 shows the spearman correlation of the four measured performance metrics to the graph metrics in a two controller scenario for Kandoo and HyperFlow, respectively. The full correlation matrix has been abbreviated for the sake of readability by filtering the metrics showing no strong correlation in either of the distributed controller architectures.

Overall, the figure details that the control plane traffic is highly correlated to the betweenness and the number of links. Whereas, a higher betweenness indicates a more consistent forwarding configuration, as more nodes are contained within the different shortest paths, an increasing number of links results in more messages sent during the topology discovery. Hence, an increase of either of these two metrics results in more traffic at the control plane. Furthermore, a loose correlation to metrics that measure the longest paths of the network can be seen, since they are also correlated to the betweenness. This is due to the fact that longer paths indicate a less inter-meshed network and hence the traffic has to be routed along these paths with no alternative routes available.

The correlation of the hit ratio is contrary to the control plane traffic and thus, has the exact opposite sign of the influences reflecting the control plane traffic. However, the correlations for the hit ratio and the graph metrics are generally stronger except the number of nodes and links, since the hit ratio is mainly influenced by the layout of the paths and less by other traffic sources like the topology discovery.

For the round-trip time there are no strong correlations visible. This is due to the fact that the geographical distances of the sites vary too greatly over all networks, due to their physical deployments.

Last, the largest difference between the two architectures is seen in the synchronization traffic. Although similar trends within the correlations exist for HyperFlow and Kandoo, i.e., the control plane traffic or hit ratio, the correlations for HyperFlow are stronger for the synchronization traffic. As the betweenness and other indicators of path lengths influence the number of packets sent by the network discovery, they also impact the amount of information which needs to be synchronized to all controller instances.

In contrast, the overall correlation for the required synchronization to the graph metrics is weaker for Kandoo. This is caused by the fact that the required amount of synchronization is mainly dependent on the switch-to-controller mappings due to the partial views of the local instances, which vary from network to network. E.g., there are networks where only one switch is assigned to the first and the rest is associated to the second controller, since the controller placements are not optimized for balanced assignments as discussed previously. Again, the correlations have the opposite sign as for the control plane traffic. In detail, networks with fewer and longer paths induce a higher synchronization overhead, since packets are more likely to leave an enclosed controller area. Therefore, the mismatches can cause more root requests, even though the overall hit ratio may be higher, depending on the topology.

*2) Principal Component Analysis (PCA):* To classify and compare the examined networks, we utilize the Principal Component Analysis (PCA) to reduce the dimensions of the graph metrics. This allows us to better visualize the networks as well as to apply clustering algorithms, hence resulting in groupings of similar networks regarding their performance and topological features. The clustering was applied to the first and second principal component, which have been obtained

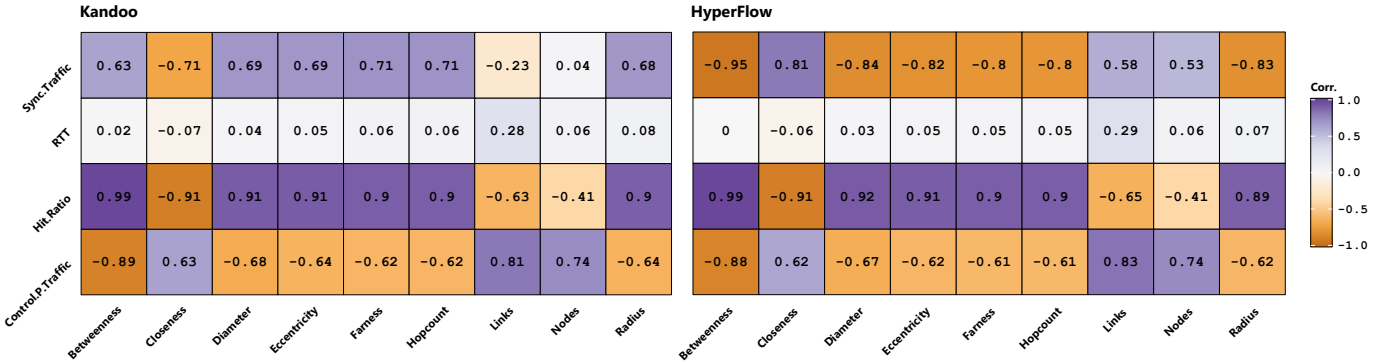



Fig. 2: Correlation of performance and graph metrics for Kandoo and HyperFlow in a two controller scenario.

by the PCA and the number of clusters was first determined by a majority vote provided by the NbClust R package[9]. The resulting final 4 clusters are detailed in the following.

Cluster 1 consists of only three networks, which reflect the most extreme topologies regarding the longest paths in the network. Here, the average diameter consists of 15.7 nodes. Since traffic always follows the same long path without alternative routes, networks within this cluster induce the least amount of control plane traffic onto the controller on average and have the highest mean hit ratio for both Kandoo and HyperFlow. In addition, these networks generate the lowest mean synchronization traffic for HyperFlow due to less traffic imposed by the topology discovery, caused by the chain-like nature of the topologies. Yet, for Kandoo these networks have the worst performance regarding the average synchronization traffic. Since the local views are the least sufficient in these networks, due to the degree of fragmentation imposed by the lack of node connectivity, the root has to be requested more frequently. The 13 networks in cluster 2 are more moderate, with respect to their path lengths and betweenness. E.g., the networks contained in this cluster have a mean diameter of around 10.5. Hence, the overall performance of networks within this cluster is more balanced and an individual metric is not severely affected in a positive or negative manner. The path lengths in the networks assigned to the clusters 3 and 4 are even shorter, with diameters of 5.9 and 6.4, respectively, which is represented by the similar position of the centroids on the first dimension. The characteristic which differentiates the networks within these two clusters is their number of nodes and links. Cluster 3 consisting of 18 networks, generally contains networks with more nodes, i.e., the average amount of switch nodes is 35.3, whereas the other clusters have a mean below 29.7. Furthermore, the intermeshing of the contained networks as well the number of links is higher with an average amount of 52 links, while the other clusters contain less than 38 links on average. Therefore, Cluster 3 performs the worst on average regarding the hit ratio and the control plane traffic, which is caused by the increased path variety. This is even more amplified by the higher number of possible target nodes for pings. For HyperFlow, this also results in more synchronization traffic due to more traffic generated by the topology discovery. For Kandoo's local instances less synchronization traffic is induced on average. Cluster 4 is formed of a total of 16 networks and contains shorter, more diverse paths on average, hence it behaves similarly to Cluster 3, resulting in a lower mean hit ratio and higher mean control plane traffic. On average, it performs the best regarding the synchronization traffic for Kandoo, as a result of the high interconnections and lower number of nodes, resulting in a lower path variety versus Cluster 3, but suitable intra-fragment views.

For the most part, the previous observations hold true for all observed controller amounts, however, the differences start to blur with more utilized controllers. Especially the influence of the synchronization traffic is reduced for more controllers, as the switch-to-controller mappings become more important. E.g., for five controllers the required synchronization in clusters 3 and 4 is almost indistinguishable for HyperFlow.

Regardless of a worsened hit ratio and higher control plane traffic, the majority of networks is contained in the clusters 3 and 4. This stems from the fact that removing links from a network generally results in a higher round-trip time, since more service times of switches are induced and a higher physical distance has to be covered.

To summarize, when designing a network in real-life, the overall goal is important as the investigated performance indicators are adversarial. If the goal is to minimize the control plane traffic and simultaneously to increase the hit ratio, routing all traffic over the same path, even if it is a long path, is beneficial. Yet, this is at the cost of the round-trip time and response times, which is disadvantageous for the Quality of Service (QoS) for users. In contrast, if the goal is to optimize for low round-trip times more interconnected networks are beneficial, yet resulting in a larger load imposed on the control plane.

### C. Performance Prediction

At last we utilize the results of the prior investigations and continue by focusing on the prediction of network performance via graph metrics, enabling the prediction of a network's performance without need for physical deployment or simulation.

The model is derived by Linear Regression (LR) and the training set is composed of artificially created networks, which

[9]https://cran.r-project.org/web/packages/NbClust/

TABLE II: Performance prediction accuracy.

| | | Kandoo | | | HyperFlow | | |
|---|---|---|---|---|---|---|---|
| | | 2C | 3C | 5C | 2C | 3C | 5C |
| MAPE | **Control P. Traffic** | 0.0538 | 0.0583 | 0.0589 | 0.0526 | 0.0593 | 0.0538 |
| | **Hit-to-Miss Ratio** | 0.0524 | 0.0509 | 0.0483 | 0.0511 | 0.0495 | 0.0506 |
| | **Round-Trip Time** | 0.6754 | 0.6832 | 0.6970 | 0.6397 | 0.6253 | 0.5663 |
| | **Sync. Traffic** | 1.3930 | 0.7299 | 0.3770 | 0.0446 | 0.0409 | 0.0353 |

aim to cover networks from all of the previously identified clusters. Hence, the challenge of obtaining sufficient data is strongly reduced. In contrast, the test set consists of the evaluated Topology Zoo networks. To evaluate the model, the four performance indicators are examined regarding the Mean Absolute Percentage Error (MAPE) of the predicted value, which are detailed in Table II.

For both distributed controller architectures, the prediction model for the control plane traffic is composed of the average betweenness and the number of links in a network. While the number of link represents the traffic caused by network topology discovery, the betweenness reflects the consistency of routing and thus, the relayed traffic resulting from mismatched packets. Here, the table shows for both architectures that the prediction is accurate. Whereas the precision fluctuates for HyperFlow for a higher number of deployed controllers, the overall prediction for Kandoo becomes less precise. However, the imprecision from three to five controllers does not change significantly. As more controllers introduce new effects that are unexplainable by the graph metrics alone, their correlation weakens and thus results in an overall more similar performance of all networks. I.e., a higher fragmentation causes the switch assignments to be more akin in terms of balance, as discussed previously, thus a single controller instance is less dominant regarding the number of managed switches.

The hit-to-miss ratio for Kandoo and HyperFlow is predicted via the betweenness only. The prediction is generally more precise than for the control plane traffic, caused by the high correlation of the betweenness and hit ratio.

Regarding the round-trip time, the model suffers from high imprecision for both architectures. As seen in the correlation analysis, none of the proposed graph metrics shows a modest or high correlation to the round-trip time. Therefore, a new metric was utilized to build the linear model, namely the average path delay of shortest paths, which includes service times and transmission delays at and between switches. Thus, it can be seen as a weighted version of the average hop count. Yet, due to the differences for the average transmission delay of the test training set, the predictions remain inaccurate. Increasing the number of controllers worsens the accuracy for Kandoo even further, as the root gains more importance which is not reflected in the prediction model. As opposed to HyperFlow, which does not feature a root instance, the prediction improves due to the more similar behavior of the networks as explained before concerning the variation for the control plane traffic.

Last, the prediction model for the synchronization traffic is examined. For Kandoo, the average closeness is the main driver of the model, since it is an indicator of how chain-like a network is and thus, how well the individual fragments are able to route the traffic within the individual controller boundaries. For HyperFlow, the model is influenced most by the betweenness and number of links, since each link creates a discovery message which needs to be synchronized. For both architectures, the accuracy increases for more deployed controllers, as the fragmentation of the network benefits the prediction of the synchronization traffic. This stems from the fact that the traffic of all networks becomes increasingly similar due to more balanced switch assignments as explained for the control plane traffic previously. Kandoo benefits even more from this effect, since the root is more likely to be involved when more controllers are deployed and therefore, all networks behave more alike in general. However, the prediction is generally worse, since Kandoo is highly dependent on the switch mappings.

To conclude, the prediction for HyperFlow is more precise with a few exceptions. The hierarchy of the distributed controller architecture Kandoo imposes an additional complexity onto the network, which is not solely explainable by the graph metrics.

## VI. Conclusion

Motivated by the increasing popularity of Software-defined Networks (SDN) and real-world deployments, we evaluated distributed controller architectures, identified Key Performance Indicators (KPIs), examined their interactions via simulation, presented a classification of network topologies based on their graph metrics as well as proposed a prediction model, which respects the entire SDN ecosystem.

We find that the hierarchical distributed controller architecture performs better regarding the induced control plane and synchronization traffic at local instances in comparison to a flat controller architecture. Yet, this comes at the cost of additional delays by requesting further information from the root controller, which is reflected in larger round-trip times. Here, the flat controller architecture is able to produce similar or better results. As the network topology has a major influence on the performance, we used the Principal Component Analysis (PCA) to reduce the dimensions of influence factors and hence applied clustering algorithms to classify a wide variety of networks into four distinct categories. Using Linear Regression (LR), we were able to accurately predict the performance of networks, regarding the control plane and synchronization traffic as well as the hit ratio of flow tables based solely on a variety of graph metrics.

In future work, we will focus on applying Machine Learning to enhance the prediction model to also incorporate non-linear correlations of the influence factors and incorporate a wider variety of UDP-based traffic patterns such as QUIC as well as verifying our results on large scale test beds.

## REFERENCES


[1] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, and et al., "B4: Experience with a Globally-Deployed Software Defined WAN," *ACM SIGCOMM Computer Communication Review*. ACM, 2013.

[2] V. Yazici, M. O. Sunay, and A. O. Ercan, "Controlling a Software-Defined Network via Distributed Controllers," *Computing Research Repository (CoRR)*, 2014.

[3] N. Gray, T. Zinner, S. Gebert, and P. Tran-Gia, "Simulation Framework for Distributed SDN-Controller Architectures in OMNeT++," in *8th EAI International Conference on Mobile Networks and Management (MONAMI 2016)*, Springer, 2016.

[4] 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*, USENIX, 2010.

[5] 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*, ACM, 2012.

[6] M. Jarschel, F. Wamser, T. Hohn, T. Zinner, and P. Tran-Gia, "SDN-based application-aware networking on the example of youtube video streaming," in *2013 Second European Workshop on Software Defined Networks*, IEEE, 2013.

[7] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks," in *Proceedings of the 2012 Asia Pacific signal and information processing association annual summit and conference*, IEEE, 2012.

[8] P. Xiong, H. Hacigumus, and J. F. Naughton, "A software-defined networking based approach for performance management of analytical queries on distributed data stores," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, ACM, 2014.

[9] C. C. Machado, L. Z. Granville, A. Schaeffer-Filho, and J. A. Wickboldt, "Towards SLA policy refinement for QoS management in software-defined networking," in *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, IEEE, 2014.

[10] A. Van Bemten, N. Đerić, 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.

[11] A. Nguyen-Ngoc, S. Lange, S. Geissler, T. Zinner, and P. Tran-Gia, "Estimating the Flow Rule Installation Time of SDN Switches When Facing Control Plane Delay," in *International Conference on Measurement, Modelling and Evaluation of Computing Systems*, Springer, 2018.

[12] Y. Zhao, L. Iannone, and M. Riguidel, "On the performance of SDN controllers: A reality check," in *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, IEEE, 2015.

[13] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, "A flexible OpenFlow-controller benchmark," in *2012 European Workshop on Software Defined Networking*, IEEE, 2012.

[14] A. Nguyen-Ngoc, S. Raffeck, S. Lange, S. Geissler, T. Zinner, and P. Tran-Gia, "Benchmarking the ONOS Controller with OFCProbe," in *2018 IEEE Seventh International Conference on Communications and Electronics (ICCE)*, IEEE, 2018.

[15] J. B. Silva, F. S. D. Silva, E. P. Neto, M. Lemos, and A. Neto, "Benchmarking of mainstream sdn controllers over open off-the-shelf software-switches," *Internet Technology Letters*, 2020.

[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)*, USENIX, 2012.

[17] E. G. Renart, E. Z. Zhang, and B. Nath, "Towards a GPU SDN controller," in *2015 International Conference and Workshops on Networked Systems (NetSys)*, IEEE, 2015.

[18] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "SDN controllers: A comparative study," in *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, IEEE, 2016.

[19] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM Computer Communication Review*. ACM, 2010.

[20] 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*, ACM, 2011.

[21] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM SIGCOMM Computer Communication Review*. ACM, 2012.

[22] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An effective approach to controller placement in software defined wide area networks," *IEEE Transactions on Network and Service Management*. IEE, 2017.

[23] 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.

[24] P. Mahadevan, D. Krioukov, M. Fomenkov, X. Dimitropoulos, K. Claffy, and A. Vahdat, "The Internet AS-level topology: three data sources and one definitive metric," *ACM SIGCOMM Computer Communication Review*. ACM, 2006.

[25] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger, "Network topology generators: Degree-based vs. structural," *ACM SIGCOMM Computer Communication Review*. ACM, 2002.

[26] J. M. Hernández and P. Van Mieghem, "Classification of graph metrics," Delft University of Technology: Mekelweg, The Netherlands, 2011.

[27] M. Yang, X. R. Li, H. Chen, and N. S. Rao, "Predicting internet end-to-end delay: an overview," in *Proceedings of the Thirty-Sixth Southeastern Symposium on System Theory*, IEEE, 2004.

[28] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an OpenFlow architecture," in *2011 23rd International Teletraffic Congress (ITC)*, IEEE, 2011.

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