Control Plane Latency With SDN Network Hypervisors: The Cost of Virtualization

Andreas Blenk, Arsany Basta, Johannes Zerwas, *Student Member, IEEE*, Martin Reisslein, *Fellow, IEEE*, and Wolfgang Kellerer, *Senior Member, IEEE*

***Abstract*—Software defined networking (SDN) network hypervisors provide the functionalities needed for virtualizing software-defined networks. Hypervisors sit logically between the multiple virtual SDN networks (vSDNs), which reside on the underlying physical SDN network infrastructure, and the corresponding tenant (vSDN) controllers. Different SDN network hypervisor architectures have mainly been explored through proof-of-concept implementations. We fundamentally advance SDN network hypervisor research by conducting a model-based analysis of SDN hypervisor architectures. Specifically, we introduce mixed integer programming formulations for four different SDN network hypervisor architectures. Our model formulations can also optimize the placement of multi-controller switches in virtualized OpenFlow-enabled SDN networks. We employ our models to quantitatively examine the optimal placement of the hypervisor instances. We compare the control plane latencies of the different SDN hypervisor architectures and quantify the cost of virtualization, i.e., the latency overhead due to virtualizing SDN networks via hypervisors. For generalization, we quantify how the hypervisor architectures behave for different network topologies. Our model formulations and the insights drawn from our evaluations inform network operators about the trade-offs of the different hypervisor architectures and help choosing an architecture according to operator demands.**

***Index Terms*—Integer linear program, network hypervisor architecture, network virtualization, software defined networking, virtual software defined network embedding.**

I. Introduction

NETWORK Virtualization (NV) enables multiple virtual networks, each specifically tailored to the demands of a particular set of network services or end-user applications, to operate on a shared underlying physical network substrate [1]. Software Defined Networking (SDN) is an emerging paradigm that introduces flexible operation and programmability into communication networks [2]. By combining NV and SDN, virtual SDN networks (vSDNs) can be created on a given physical SDN network. Tenants can program their vSDN resources via open network interfaces and protocols, e.g., OpenFlow [2], and run their own vSDN controllers. For instance, for Network Function Virtualization (NFV), vSDNs can be used to flexibly interconnect virtual network functions and to control their network traffic via SDN [1].

To allow multiple network operating systems to run in parallel, so called SDN network virtualization hypervisors have been introduced [3]–[5]. SDN network virtualization hypervisors [6], which we refer to as *hypervisors* for brevity, operate as an intermediate layer between SDN network infrastructures and vSDN controllers. SDN network hypervisors present the vSDN controllers with virtual SDN networks (vSDNs), which are composed of virtual SDN switches. The vSDN controllers are connected via the hypervisors to their vSDN switches (see Fig. 1(a)). As hypervisors operate transparently to vSDN controllers, each vSDN controller only sees its corresponding vSDN switches. Accordingly, hypervisors do not limit tenants to the application-controller interfaces provided by traditional SDN controllers, e.g., ONOS [7] or OpenDaylight [8]. With hypervisors, tenants can still use conventional SDN network interfaces/protocols, e.g., OpenFlow, to control their vSDNs. Thus, as tenants are not limited to special implementations, they can choose freely from all available SDN controller implementations and extend them according to their needs.

In SDN networks, good control plane performance, such as low control plane latency, is important for achieving high network performance. For instance, high control plane latencies may lead to long flow set-up times, which are detrimental for many services, e.g., for DNS requests. In non-virtualized SDN networks, the Controller Placement Problem (CPP) tackles the question of how many controllers are needed and where to place them in the network in order to achieve a high network performance. While SDN controllers connect directly to the SDN infrastructure, hypervisors serve as controllers to the underlying substrate network in virtualized SDN networks. As the paths between tenant controllers and the vSDNs have to traverse the hypervisor instances, tenants may experience longer controller to switch connections. These longer paths introduce control plane latency overhead, which we call the *cost of virtualization*. As hypervisors are mostly implemented in software, they can be flexibly placed in the network, e.g., at data center locations. Efficient virtualization of SDN networks requires sophisticated techniques for placing

Manuscript received February 15, 2016; revised May 11, 2016, June 14, 2016, and June 20, 2016; accepted June 30, 2016. Date of publication July 7, 2016; date of current version September 30, 2016. This work is part of a project that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation program (grant agreement No 647158 - FlexNets) and from the A. von Humboldt Foundation through an F.W. Bessel Research Award. The associate editor coordinating the review of this paper and approving it for publication was P. Chemouil. *(Corresponding author: Martin Reisslein.)*

A. Blenk, A. Basta, J. Zerwas, and W. Kellerer are with the Chair of Communication Networks, Technische Universität München, Munich 80290, Germany (e-mail: andreas.blenk@tum.de; arsany.basta@tum.de; johannes.zerwas@tum.de; wolfgang.kellerer@tum.de).

M. Reisslein is with the Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ 85287 USA (e-mail: reisslein@asu.edu).

Digital Object Identifier 10.1109/TNSM.2016.2587900

hypervisor instances in the network. Only proper hypervisor placement provides vSDN tenants with the best possible performance. We call this the *k*-Network Hypervisor Placement Problem (*k*-HPP) in this article. The *k*-HPP answers the question of how many hypervisor instances *k* are needed and where the hypervisor instances should be placed in the network.

While some hypervisor architectures rely only on basic SDN features, some hypervisors can make use of special switch functionalities, e.g., the functionality to support multiple controllers, the so-called multi-controller feature. Multi-controller switches can simultaneously connect to multiple SDN controllers, i.e., multiple hypervisor instances. Multi-controller switches may improve control plane performance, e.g., reduce control plane latency. However, multi-controller switches may demand additional synchronization between distributed hypervisor instances. For instance, hypervisor instances may need to synchronize flow table access or to carefully plan the allocation of available flow table space. Thus, the placement of multi-controller switches needs to be carefully planned. We refer to this planning problem as the Multi-controller Switch Deployment Problem (McSDP) in this article.

SDN network virtualization hypervisors can be implemented and operated in either a centralized ($k = 1$) or distributed ($k > 1$) manner [6]. Due to the variety of existing hypervisor architectures and their ability to make use of special network functionalities, the *k*-HPP cannot simply be solved by referring to solutions of the SDN Controller Placement Problem (CPP) [9]. The *k*-HPP is fundamentally different from the CPP due to the following aspects: (1) the existence of multiple vSDNs with individual demands, e.g., for control plane latency; (2) the functionality of hypervisor instances to serve as intermediate nodes between multiple vSDN controllers and the underlying physical SDN network, i.e., the SDN network to controller connections need to traverse the hypervisor instances; (3) the ability of hypervisor architectures to make use of the multi-controller feature of SDN nodes (switches) for minimizing control plane latency.

Our main contribution in this article is the in-depth study of the fundamentally new *k*-HPP for four different SDN network hypervisor architectures with respect to control plane latency. We provide mathematical mixed integer programming models for the four architectures. Our models jointly solve the McSDP and the k-HPP. We investigate the determination of the best locations of hypervisor instances and multi-controller switches with our models for real network topologies and a wide range of vSDN requests. We analyze the trade-offs among four hypervisor latency objective metrics. We also closely examine the impact of virtualization on the individual SDN network requests. Furthermore, we analyze the benefits of a priori optimization of the locations of the vSDN controllers. Specifically, we investigate the impacts of three different controller placement strategies on the *k*-HPP and McSDP. The current study substantially extends the preliminary conference paper [10] which presented results for the placement of a single hypervisor instance ($k = 1$) for single-controller switches only; in contrast, we examine in detail the general *k*-HPP with multi-controller switches in this paper.

The remainder of this paper is structured as follows. The needed background and an overview of related work are presented in Section II. In Section III, we introduce the four SDN network hypervisor architectures, which we examine in depth in this paper. In Section IV, we provide mathematical formulations of the *k*-HPP and the McSDP. In Section V, we provide mathematical models to solve the *k*-HPP and McSDP based on mixed integer programming. The evaluation setup is explained in Section VI, while results are presented in Section VII. Conclusions and future work are outlined in Section VIII.

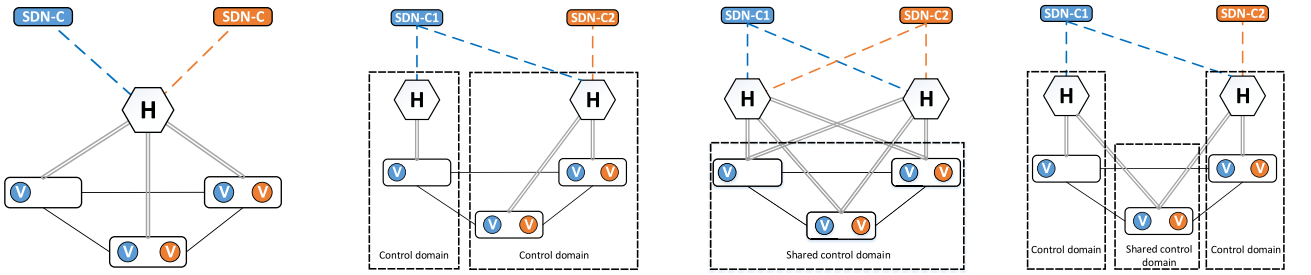
II. Background & Related Work

A. Background

1) Software Defined Networking & Multiple Controllers Feature: Software Defined Networking (SDN) decouples the control plane from the data plane of forwarding hardware, e.g., routers or switches. The control plane runs logically centralized in SDN controllers. SDN controllers run in software, thus, can be flexibly deployed on commodity hardware, i.e., servers. OpenFlow [11] is one protocol that enables the communication between SDN controllers and the networking hardware, i.e., SDN switches.

OpenFlow 1.2 [12] introduced and defined the multiple controllers feature. The multiple controllers feature allows switches to simultaneously connect to multiple SDN controllers. In non-virtualized SDN networks, the feature can be used for controller fail-over or load balancing. The number of controllers that a given switch simultaneously connects to may be limited [13]. The OpenFlow specification [12] defines an `OFPCR_ROLE_EQUAL` mode, in which all connected controllers can fully access and control the switch resources. The `OFPCR_ROLE_EQUAL` mode requires the SDN controllers to synchronize the management of the switch resources. In this article, we analyze how the multiple controllers feature can be used to reduce the control plane latency of vSDNs. Specifically, we distinguish between single-controller switches, which can connect to one SDN controller (hypervisor instance) at a time, and multi-controller switches, which use the multiple controllers feature to connect simultaneously to multiple SDN controllers (hypervisor instances).

2) SDN Network Hypervisors: SDN network hypervisors [6] sit between vSDN controllers and the underlying physical SDN network, as illustrated in Fig. 1(a). Similar to SDN controllers, they are mostly implemented in software. Each hypervisor instance implements the entire virtualization stack. That is, each hypervisor instance can virtualize a part of the underlying physical SDN network. Distributed hypervisor instances may need to synchronize their states, e.g., for load balancing purposes. The impact of the synchronization load is outside the scope of this article, and is a direction for future work. In general, a hypervisor instance provides the following virtualization functions: abstraction (virtualization), translation, and isolation [6]. Hypervisors abstract the underlying physical SDN network, i.e., they provide all necessary information for operation to the vSDN controllers, e.g., topology information. Tenant controllers need to connect to



(a) Centralized architecture, $k = 1$

(b) Distributed architecture + single-contr. SDN switches, $k > 1, M = 0$

(c) Distributed architecture + multi-controller SDN switches, $k > 1, M = |\mathcal{V}|$

(d) Distr. hybrid arch. with single- and multi-contr. SDN switches, $k > 1, 0 < M < |\mathcal{V}|$

Fig. 1. Illustration of four hypervisor architecture categories (characterized by number of hypervisor instances k and number of multi-controller switches M) for an example SDN network with two virtual SDN networks (vSDNs). The blue and orange color differentiate the two vSDNs. A hypervisor instance (location) is represented by a hexagon. The square dashed boxes represent the control domains in case of multiple hypervisor instances. A circle, labeled with "V", is a vSDN switch (node) hosted on a larger box with rounded edges, which represents a physical SDN switch (node). The solid lines between these boxes represent the data plane connections, i.e., the edges of the physical SDN network. A dashed line represents a connection between an SDN controller (SDN-C) and a hypervisor instance "H". A double solid line represents a physical connection between a hypervisor and a physical SDN switch.

hypervisor instances to access their virtual network resources, i.e., virtual SDN switches. Further, a vSDN controller can connect to multiple hypervisor instances. Since all tenant control traffic has to pass through hypervisor instances, the hypervisor instances become a critical component of vSDNs.

B. Related Work

We review main research areas related to the virtualization of SDN networks in this section and distinguish our present study on SDN hypervisor placement from related work.

1) Facility Location Problem: As indicated by [9], the general facility location problem (FLP) is the general problem behind the SDN controller placement problem. Similarly, the k-HPP can be related to the hierarchical facility location problem. The task of the hierarchical facility location problem is to find the best facility locations in a multi-level network. The facilities at higher levels have to serve the facilities at lower levels, while customers need to be served at the lowest level. A similar layering can be applied to the k-HPP. Tenant controllers need to connect to hypervisor instances, while hypervisor instances need to connect to SDN switches at the lowest level. Different variations, adaptations to real problems, and overviews of the FLP are provided in [14]–[18]. The unique feature of the k-HPP is the differentiation of groups of customers, i.e., individual vSDNs, which need to be specifically operated by their corresponding tenant controllers.

2) SDN Controller Placement: The SDN Controller Placement Problem (CPP) for non-virtualized SDN networks has been initiated in [9]. The CPP targets the question of how many controllers are needed and where to place them. Using a brute-force method, [9] evaluated the impact of controller placement on average and maximum latency metrics for real network topologies. The authors concluded that five controllers are sufficient to achieve an acceptable control plane latency for most topologies. As different optimization objectives, e.g., load and delay, are critical for the operation of SDN networks, multi-objective optimization approaches have been applied to the CPP [19]. The framework in [19] uses simulated annealing to analyze the CPP for different network topologies with respect to multiple objectives, e.g., latency and resilience. As real SDN networks have node and link capacity constraints, mathematical models for solving the CPP with node and link capacity have been studied in [20] and [21]. Considering capacity constraints during planning protects SDN controllers from overload situations. Distributed SDN controllers can be organized in a hierarchy to achieve resilience [22]. The study [22] provides an algorithm and performance comparisons for k-center and k-median-based algorithms. Further CPP research either considers different metrics, e.g., resilience or load balancing [23]–[25], or incorporates different methodologies, e.g., clustering. A dynamic version of the CPP, where the rate of flow setups varies over time, has been studied in [26].

In the present study, we solve the CPP a priori for maximum or average latency objectives and use the CPP solution as an input to our optimization. This two step optimization allows us to analyze the impact of the vSDN controller placement on the hypervisor placement.

3) Virtual Network Embedding: The embedding of virtual to physical network resources is an integral part of network virtualization. There are many algorithms to solve the VNE problem [27]. Some VNE algorithms consider technology aspects of the infrastructure, e.g., flexible path splitting [28]. In general, VNE research mostly neglects the control part when virtualizing networks, i.e., neglects the connections from the tenants to their resources. Only a few studies have incorporated the control (node) while embedding the virtual networks. For instance, [29] uses heuristic algorithms (greedy, simulated annealing) to optimize the vSDN embedding for a balanced load or for latency. The embedding also considers the impact of placing the SDN controller; however, the hypervisor instances are *not* taken into account while embedding the virtual resources. Generally, when not considering the control plane, existing VNE algorithms can be directly applied to efficiently solve the mapping of vSDN resources [28], [30]. However, to the best of our knowledge, the VNE research to date has *not* incorporated the full design and optimization of

the control plane, i.e., the controller and hypervisor embedding, which is particularly important for virtual SDN network embedding.

III. SDN Network Hypervisor Architectures

In this section, we introduce four hypervisor architecture categories. We categorize the architectures into *centralized architectures* and *distributed architectures*. We further sub-classify the distributed architectures into architectures operating with *single-controller SDN switches* or with *multi-controller SDN switches*. In addition, we consider distributed *hybrid architectures* that combine single- and multi-controller SDN switches. A single centralized hypervisor instance (at a single location) provides the virtualization functionality in a centralized architecture. In contrast, in a distributed hypervisor architecture, multiple hypervisor instances that are distributed over multiple locations realize the virtualization functionality. We denote the number of hypervisor instances by k and the number of multi-controller switches by M.

A. Centralized Network Hypervisor Architecture

The centralized SDN network hypervisor architecture ($k = 1$) deploys only a single hypervisor instance (at a single location) for SDN network virtualization. Virtual SDNs can be provided by running this single hypervisor instance at one physical network location. FlowVisor [3] is an example of a centralized hypervisor architecture. In this article, the centralized hypervisor architecture works with SDN switches (network elements) compliant with the OpenFlow specification [12]. OpenFlow specification [12] compliant SDN switches do not provide any specialized functionalities to support virtualization. In case a virtualization functionality cannot be provided by OpenFlow compliant switches, the hypervisor has to provide the functionality. This implies that special virtualization functionalities need to be implemented outside the OpenFlow switch domain.

Fig. 1(a) shows an exemplary centralized hypervisor architecture set-up. The hypervisor instance connects down to three physical SDN switches (nodes, network elements) and up to two vSDN controllers. The upper left physical SDN switch provides a virtual switch for the left tenant. The upper right and lower middle physical switches host two virtual switch instances for each tenant. The single centralized hypervisor instance is the SDN controller of all physical SDN switches. All control traffic of the vSDNs has to pass through this single hypervisor instance. From the switches, the hypervisor forwards the control traffic towards the corresponding vSDN controller. The control plane latency of such centralized SDN hypervisor architecture has already been modeled and analyzed via simulations in [10]. In this paper, we compare the centralized architecture to the distributed architectures and additionally investigate the impact of the network topology.

B. Distributed Network Hypervisor Architecture for Single-Controller SDN Switches

For scalability reasons, a hypervisor can be distributed into multiple (k, $k > 1$) hypervisor instances that are distributed over multiple (k) locations in the network. Suppose that the SDN switches can only connect to one hypervisor instance at a time ($M = 0$). Accordingly, the physical SDN network is split into multiple control domains, whereby one hypervisor instance is responsible for a given domain. An example for a distributed SDN hypervisor architecture operating with single-controller SDN switches is FlowN [31].

An example distributed architecture with two hypervisor instances is illustrated in Fig. 1(b). The SDN switches are controlled by $k = 2$ hypervisors. The left hypervisor instance controls the upper left SDN switch, while the right hypervisor instance controls the other SDN switches. Accordingly, the SDN switches are split into two distinct control domains. Each SDN switch connects to either one of the $k = 2$ hypervisor instances. Note that one hypervisor instance can connect to multiple controllers (as illustrated for the right hypervisor instance). As the virtual switch instances of the left SDN controller 1 (colored in blue) are in different control domains of the hypervisors, SDN controller 1 connects simultaneously to two hypervisor instances.

C. Distributed Network Hypervisor Architecture for Multi-Controller SDN Switches

The distributed network hypervisor architecture for multi-controller switches realizes the SDN virtualization via multiple separated hypervisor instances ($k > 1$), similar to Section III-B. However, all $|\mathcal{V}|$ physical SDN switches can now simultaneously connect to multiple hypervisor instances as all switches support multiple controllers (i.e., $M = |\mathcal{V}|$). As a result, there is no separation of the control domain of the SDN switches as each switch can be simultaneously controlled by multiple hypervisor instances. An example for the distributed hypervisor architecture with multi-controller SDN switches is DITRA [32]. With DITRA, a given physical SDN switch can simultaneously connect to multiple hypervisor instances. DITRA [32] operates with legacy SDN switches that support the multi-controller feature, as it was introduced with OpenFlow version 1.2 [12]. That is, DITRA does not require extensions of the switch hardware.

While each physical SDN switch is only connected to a single hypervisor instance in Fig. 1(b), Fig. 1(c) shows two hypervisor control connections for each physical SDN switch. The multi-controller feature allows an SDN switch to connect to multiple different hypervisor instances during operation. However, as switch resources, e.g., switch CPU and flow tables, are shared and not strictly isolated, coordination between the different hypervisor instances may be necessary.

D. Distributed Hybrid Network Hypervisor Architecture

In general, it may not be necessary for all SDN switches to support the multiple controllers feature to achieve a specific optimization objective. Furthermore, due to the hardware limitations for supporting multiple controllers and the additional coordination overhead, thorough planning of an SDN network is important. The result of such planning could be that only some of the switches implement or use the multi-controller feature, while others are supporting or operating

only the single-controller mode. This leads to the fourth hypervisor architecture category, namely a distributed architecture that operates on hybrid SDN networks. We define a hybrid SDN network as an SDN network that simultaneously uses single-controller and multi-controller SDN switches.

Fig. 1(d) illustrates an example of the distributed hybrid architecture. While the upper left switch connects only to the left hypervisor instance and the upper right switch connects only to the second hypervisor instance, the lower middle switch ($M = 1$) connects to both hypervisor instances. Thus, the control domain of the lower middle switch is shared by both hypervisor instances. We can separate the shared and non-shared control domains, as illustrated in Fig. 1(d). The switches of the non-shared control domains operate in single-controller mode, i.e., they connect to only one hypervisor instance. Specifically, our model provides the capability to prescribe a maximum permissible number M of multi-controller SDN switches for a given network topology. The solution of our optimization problem formulation provides the optimal number of multi-controller SDN switches and the separation of the SDN network into different control domains.

IV. Problem Setting for k-Network Hypervisor Placement Problem (k-HPP) and Multi-Controller Switch Deployment Problem (McSDP)

The k-Network Hypervisor Placement Problem (k-HPP) extends the Network Hypervisor Placement Problem (HPP), where only $k = 1$ network hypervisor is placed to connect virtual data plane switches to their corresponding vSDN controllers. We also introduce the Multi-controller Switch Deployment Problem (McSDP), which determines the number and the locations of multi-controller enabled switches. This section first introduces the setting for these problems by defining the notation for the physical SDN network and the vSDN requests. Then, we introduce the mathematical definition of the k-HPP and the McSDP.

A. Network Models

The input of the k-Network Hypervisor Placement Problem is given by the set of vSDN requests $\mathcal{R}$, which are to be fulfilled with a given physical SDN network graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$.

1) Physical SDN Network Specification: Table I summarizes the notation for the physical SDN network. The network is modeled as a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with physical SDN switches (network nodes) $v \in \mathcal{V}$ connected by undirected edges $e \in \mathcal{E}$. The potential hypervisor nodes (locations) are given by the set Φ. They are a subset of $\mathcal{V}$, i.e., $\Phi \subseteq \mathcal{V}$. The latency $\lambda(e)$ of an edge e is computed from the geographical distance between the two network nodes that are connected via edge e (the transmission bit rate (capacity) of the edge is not considered). The edge latency $\lambda(e)$ is used for evaluating the latency of network paths. The set $\mathcal{P}$ contains the shortest paths of the network between any network node pair. A shortest path is denoted as $(s, t) \in \mathcal{P}$. The distance, i.e., the latency, of a shortest path is denoted by $d(s, t)$. Furthermore, the function $d(s, v, t)$ gives the latency of the shortest path connection between nodes s and t via node v. This value is calculated as the sum of $d(s, v)$ and $d(v, t)$.

TABLE I
Notation for Physical SDN Network $\mathcal{G}$

Notation	Description
$\mathcal{G}(\mathcal{V}, \mathcal{E})$	Physical SDN network graph
$\mathcal{V}$	Set of physical SDN switches (network nodes), i.e., node locations
v	Physical SDN switch (network node) $v \in \mathcal{V}$
$\mathcal{E}$	Set of physical network edges
e	Physical edge $e \in \mathcal{E}$
Φ	Set of potential hypervisor nodes (locations) with $\Phi \subseteq \mathcal{V}$
$\lambda(e)$	Latency of edge e, with $\lambda(e) \in \mathbb{R}^+$
$\mathcal{P}$	Set of pre-calculated shortest paths between all network node pairs
(s, t)	Shortest path between two network nodes s and t, with $(s, t) \in \mathcal{P}$
$d(s, t)$	Latency (distance) of shortest path $(s, t) \in \mathcal{P}$
$d(s, v, t)$	Latency (distance) of path connecting nodes s and t while traversing (passing) node v.

TABLE II
Notation for Virtual SDN Network (vSDN) Requests $\mathcal{R}$

Notation	Description
$\mathcal{R}$	Set of vSDN requests
r	Virtual network request $r \in \mathcal{R}$
$\mathcal{V}^r$	Set of virtual nodes of vSDN request r, $r \in \mathcal{R}$
v^r	Virtual network node $v^r \in \mathcal{V}^r$
$\pi(v^r)$	Mapping from virtual node v^r to its physical host switch (network node) v, i.e., $\forall r \in \mathcal{R} : \pi(v^r) : \mathcal{V}^r \rightarrow \mathcal{V}$
c^r	Virtual controller node of vSDN request r, $r \in \mathcal{R}$, with $\pi(c^r) \in \mathcal{V}$

TABLE III
Problem Input for k-HPP and McSDP

Notation	Description
$\mathcal{G}$	Physical SDN network
$\mathcal{R}$	Set of virtual SDN network (vSDN) requests
k	Number of hypervisor nodes to be placed
M	Number of physical SDN switches (network nodes) supporting multiple controllers

2) Virtual SDN Network (vSDN) Request: Table II summarizes the notation for the vSDN requests $\mathcal{R}$. A vSDN request $r \in \mathcal{R}$ is defined by the set of virtual SDN network nodes $\mathcal{V}^r$ and the vSDN controller c^r. The physical SDN switch of a vSDN network node is given by the function $\pi(v^r)$, i.e., $\pi(v^r) \in \mathcal{V}$. All vSDN network nodes need to be connected to their controller instance c^r. The location of the controller is also chosen among the available network node locations, i.e., $\pi(c^r) \in \mathcal{V}$. Note that we assume a vSDN to operate only one SDN controller in this paper, i.e., we do not consider multiple SDN controllers for a given vSDN.

B. k-Hypervisor Placement Problem (k-HPP)

Table III specifies the input of the k-HPP. For a given physical SDN network $\mathcal{G}$ and set of vSDN requests $\mathcal{R}$, a prescribed number k of hypervisor locations need to be chosen among all potential hypervisor locations Φ. The result of such an optimization problem is the set of selected hypervisor locations $\mathcal{H}$. The set $\mathcal{H}$ specifies the hypervisor locations on the network, i.e., the locations where the hypervisors are actually placed.

TABLE IV
BINARY DECISION VARIABLES FOR k-HPP AND MCSDP

Notation	Description
$x_{\mathcal{H}}(h)$	=1, if a hypervisor is placed at potential hypervisor node (location) $h \in \Phi$; 0, otherwise
$x_{\mathcal{R}}(v^r, h, c^r)$	=1, if vSDN node $v^r \in \mathcal{V}^r$ is connected to controller c^r via hypervisor node (location) $h \in \Phi$; 0, otherwise
$x_{\mathcal{V},\mathcal{H}}(v, h)$	=1, if physical SDN node $v \in \mathcal{V}$ is controlled by hypervisor node $h \in \Phi$; 0, otherwise
$x_{\mathcal{M}}(v)$	=1, if physical SDN node $v \in \mathcal{V}$ is controlled by multiple hypervisor instances, i.e., if multiple node to hypervisor (controller) connections exist; 0, otherwise.

In real networks, those hypervisor locations could be data center locations, which are connected to the network topology at given network locations $v \in \mathcal{V}$.

C. Multi-Controller Switch Deployment Problem (McSDP)

We denote M for the number of multi-controller SDN network nodes. We note that in our problem formulation, we do not specify which physical SDN switches specifically support the multi-controller feature. Instead, solving our problem formulation determines which switches should support multiple controllers (hypervisors). An alternative input setting of our problem formulation could include a predetermined set of switches supporting the special multi-controller feature. In case $M = 0$, no physical SDN switch supports the multi-controller feature, i.e., no SDN switch can simultaneously connect to multiple hypervisor instances. For $0 < M < |\mathcal{V}|$, a subset of the physical SDN switches supports multiple controllers. In case $M = |\mathcal{V}|$, all physical SDN switches support multiple controllers.

V. MIXED INTEGER PROGRAMMING FORMULATION FOR k-HPP AND MCSDP

A. Decision Variables

Table IV specifies the binary decision variables of the mixed integer programming formulation of the k-HPP and McSDP. The variable $x_{\mathcal{H}}(v)$ determines whether a hypervisor is located at the network node (location) $v \in \Phi$. Note that after having solved the model, the variables $x_{\mathcal{H}}(v)$ specify the set $\mathcal{H}$ of hypervisor nodes, specifically, $\mathcal{H} = \{v \in \Phi : x_{\mathcal{H}}(v) = 1\}$. For a request $r \in \mathcal{R}$, the variable $x_{\mathcal{R}}(v^r, h, c^r)$ is set to one if the vSDN node $v^r \in \mathcal{V}^r$ is connected to the vSDN controller c^r via the hypervisor node (location) $h \in \Phi$. Note that if a path $x_{\mathcal{R}}(v^r, h, c^r)$ is set to one, then a hypervisor needs to be placed at the potential hypervisor node (location) h. The variable $x_{\mathcal{V},\mathcal{H}}(v, h)$ indicates whether physical node $v \in \mathcal{V}$ is controlled by the hypervisor instance placed at location $h \in \Phi$. The variable $x_{\mathcal{M}}(v)$ indicates whether the multi-controller feature is deployed and used at physical node $v \in \mathcal{V}$. In case of a multi-controller SDN switch, i.e., where $x_{\mathcal{M}}(v) = 1$, the variable $x_{\mathcal{V},\mathcal{H}}(v, h)$ for a given node $v \in \mathcal{V}$ is possibly one for multiple hypervisor nodes (locations) $h \in \Phi$.

B. Objective Functions

We focus on objective functions that seek to minimize the control plane latency. In particular, we introduce four latency metrics, namely maximum latency $L_{\max}$, average latency L_{avg}, average maximum latency L_{avgmax}, and maximum average latency L_{maxavg}. Average and maximum latency are traditional metrics from the related SDN controller placement problem; we extend these metrics for the k-HPP. Note that when optimizing for $L_{\max}$, L_{avgmax}, and L_{maxavg}, additional variables and constraints are needed. These variables and constraints are subsequently introduced when the metrics are presented. As these variables and constraints are objective specific, they are not described in the general constraints Section V-C. We investigate a model without capacity constraints in this study. The incorporation of capacity constraints, such as data rate and node capacity (e.g., CPU or memory capacity) are planned for future work.

1) Maximum Latency: The maximum latency for a considered hypervisor placement is the maximum latency of all utilized shortest paths from all requests $r \in \mathcal{R}$. Recall that the binary decision variable $x_{\mathcal{R}}(v^r, h, c^r)$ indicates (i.e., is equal to one) when the path from v^r via h to c^r is used. Thus, the maximum latency of all paths that have been selected to fulfill the requests $r \in \mathcal{R}$ is given by

$$L_{\max} = \max_{r \in \mathcal{R},\ v^r \in \mathcal{V}^r,\ h \in \Phi} x_{\mathcal{R}}(v^r, h, c^r) d\big(\pi\left(v^r\right), h, \pi\left(c^r\right)\big). \quad (1)$$

Minimizing the latency metric $L_{\max}$ involves minimizing a maximum over sets, which is not directly amenable to some solvers. The maximum over sets can be readily expressed as an equivalent constrained minimization problem. Specifically, we can equivalently minimize $L_{\max}$ defined through the constraints

$$L_{\max} \geq x_{\mathcal{R}}(v^r, h, c^r) d\big(\pi\left(v^r\right), h, \pi\left(c^r\right)\big), \quad \forall r \in \mathcal{R},\ \forall v^r \in \mathcal{V}^r,\ \forall h \in \Phi. \quad (2)$$

The resulting objective function is

$$\min L_{\max}. \quad (3)$$

2) Average Latency: The average latency is the average of all path latencies of all vSDNs that connect the virtual network nodes with the vSDN controllers of the respective vSDNs. For a vSDN request r, there are $|\mathcal{V}^r|$ vSDN nodes that need to be connected to the vSDN controller c^r. Thus, for a set of requests $\mathcal{R}$, there are overall $\sum_{r \in \mathcal{R}} |\mathcal{V}^r|$ paths and the average latency is

$$L_{\text{avg}} = \frac{1}{\sum_{r \in \mathcal{R}} |\mathcal{V}^r|} \sum_{r \in \mathcal{R}} \sum_{v^r \in \mathcal{V}^r} \sum_{h \in \Phi} x_{\mathcal{R}}(v^r, h, c^r) \times d\big(\pi\left(v^r\right), h, \pi\left(c^r\right)\big). \quad (4)$$

Note that this metric does not differentiate between the vSDNs. Here, no additional variable or constraint are needed, thus the average latency objective function is

$$\min\ L_{\text{avg}}. \quad (5)$$

3) Average Maximum Latency: The average maximum latency for a given hypervisor placement is defined as the average of the maximum latencies for the individual vSDN requests $r \in \mathcal{R}$. First, the maximum path latency for each vSDN request r is evaluated. Second, the average of all maximum path values is evaluated, i.e., the sum of the

maximum path latencies is divided by the total number of vSDN requests $|\mathcal{R}|$.

$$L_{\text{avgmax}} = \frac{1}{|\mathcal{R}|}\sum_{r\in\mathcal{R}}\max_{v^r\in\mathcal{V}^r,\ h\in\Phi} x_{\mathcal{R}}(v^r,h,c^r)d\big(\pi\left(v^r\right),h,\pi\left(c^r\right)\big). \quad (6)$$

In order to circumvent the maxima over sets, we define constraints for the maximum latency of each given vSDN request $r\in\mathcal{R}$:

$$L^r_{\max} \geq x_{\mathcal{R}}(v^r,h,c^r)d\big(\pi\left(v^r\right),h,\pi\left(c^r\right)\big),\ \ \forall v^r\in\mathcal{V}^r,\ \ \forall h\in\Phi. \quad (7)$$

The objective function then minimizes the average of the $L^r_{\max}$ over all requests $|\mathcal{R}|$:

$$\min\frac{1}{|\mathcal{R}|}\sum_{r\in\mathcal{R}}L^r_{\max}. \quad (8)$$

This objective function provides a relaxed average latency towards a better maximum latency per vSDN. Note that this objective function differentiates between vSDNs.

4) Maximum Average Latency: The maximum average latency is defined as the maximum of the average latencies for the individual vSDNs. First, the average latency of each requested vSDN request $r\in\mathcal{R}$ is determined. Second, the maximum of these averages is evaluated, i.e.,

$$L_{\text{maxavg}} = \max_{r\in\mathcal{R}}\frac{1}{\left|\mathcal{V}^r\right|}\sum_{v^r\in\mathcal{V}^r}\sum_{h\in\Phi}x_{\mathcal{R}}(v^r,h,c^r) \times d\big(\pi\left(v^r\right),h,\pi\left(c^r\right)\big). \quad (9)$$

This metric corresponds to the maximum of the vSDN average latencies, i.e., the maximum latencies are relaxed per vSDN towards a better overall maximum average latency. Minimizing the maximum over the set $\mathcal{R}$ is equivalent to minimizing L_{maxavg} defined through the constraints

$$L_{\text{maxavg}} \geq \frac{1}{\left|\mathcal{V}^r\right|}\sum_{v^r\in\mathcal{V}^r}\sum_{h\in\Phi}x_{\mathcal{R}}(v^r,h,c^r)d\big(\pi\left(v^r\right),h,\pi\left(c^r\right)\big) \quad \forall r\in\mathcal{R}. \quad (10)$$

The objective function then minimizes L_{maxavg}:

$$\min L_{\text{maxavg}}. \quad (11)$$

C. Constraints

We proceed to introduce the constraints for the k-HPP and McSDP.

1) Hypervisor Selection Constraint: We ensure that the number of placed hypervisor instances (i.e., the number of selected hypervisor nodes (locations)) is equal to k:

$$\sum_{h\in\Phi}x_H(h) = k. \quad (12)$$

2) Virtual Node Path Selection Constraint: Each virtual node $v^r\in\mathcal{V}$ of each vSDN request $r\in\mathcal{R}$ must be connected to its corresponding controller c^r via exactly one hypervisor node h. This means that per virtual node v^r per request r, exactly one path has to be used:

$$\sum_{h\in\Phi}x_{\mathcal{R}}(v^r,h,c^r) = 1, \quad \forall r\in\mathcal{R}\ ,\forall v^r\in\mathcal{V}^r. \quad (13)$$

3) Hypervisor Installation Constraint: We place (install) a hypervisor instance at location h (i.e., set $x_{\mathcal{H}}(h)=1$) if at least one virtual node v^r is connected to its controller c^r via the hypervisor location h (i.e., if $x_{\mathcal{R}}(v^r,h,c^r)=1$). At the same time, at most $\sum_{r\in\mathcal{R}}|\mathcal{V}^r|$ virtual nodes can be connected via a given hypervisor location h to their respective controllers. Thus,

$$\sum_{r\in\mathcal{R}}\sum_{v^r\in\mathcal{V}^r}x_{\mathcal{R}}(v^r,h,c^r) \leq x_{\mathcal{H}}(h)\sum_{r\in\mathcal{R}}|V^r|, \quad \forall h\in\Phi. \quad (14)$$

4) Physical Node to Hypervisor Assignment Constraint: We let a hypervisor node (location) h control a physical SDN switch (network node) v, if a path is selected to connect a virtual node v^r to its controller c^r via h (i.e., if $x_{\mathcal{R}}(v^r,h,c^r)=1$) and additionally, this virtual node is hosted on v, i.e., $\pi(v^r)=v$. Thus:

$$x_{\mathcal{R}}(v^r,h,c^r) \leq x_{\mathcal{V},\mathcal{H}}(\pi\left(v^r\right),h), \quad \forall r\in\mathcal{R},\ \forall v^r\in\mathcal{V}^r,\ \forall h\in\Phi. \quad (15)$$

5) Multiple Hypervisors Constraint: We determine the physical SDN switches $v\in\mathcal{V}$ that can be controlled by multiple hypervisors, i.e., the switches v (with $x_{\mathcal{M}}(v)=1$) that support multiple controllers. For a given physical multi-controller SDN switch $v\in\mathcal{V}$ (with $x_{\mathcal{M}}(v)=1$), the number of controlling hypervisors must be less than or equal to the total number of hypervisor nodes k, if the switch hosts at least one virtual SDN switch (which needs to be connected to its controller). On the other hand, for a physical single-controller SDN switch $v\in\mathcal{V}$ (with $x_{\mathcal{M}}(v)=0$), the number of controlling hypervisors must equal one, if the switch hosts at least one virtual SDN switch. Thus, for an arbitrary physical SDN switch (node) $v\in\mathcal{V}$ (irrespective of whether v is a single- or multi-controller SDN switch), the total number of controlling hypervisor instances (locations) must be less than or equal to $[1-x_{\mathcal{M}}(v)]+kx_{\mathcal{M}}(v)$. Thus,

$$\sum_{h\in\Phi}x_{\mathcal{V},\mathcal{H}}(v,h) \leq \left[1-x_{\mathcal{M}}(v)\right]+kx_{\mathcal{M}}(v), \quad \forall v\in\mathcal{V}. \quad (16)$$

We note that some solvers may unnecessarily set some $x_{\mathcal{V},\mathcal{H}}(v,h)$ to one for a hypervisor node h, even though network node v does *not* host any virtual node v^r that is connected to its corresponding controller c^r via hypervisor node h. This is because the solver can find a valid minimal latency solution while setting some $x_{\mathcal{V},\mathcal{H}}(v,h)$ unnecessarily to one. We circumvent this issue by forcing $x_{\mathcal{V},\mathcal{H}}(v,h)$ to zero if no corresponding path for this hypervisor instance was selected:

$$x_{\mathcal{V},\mathcal{H}}(v,h) \leq \sum_{r\in\mathcal{R}}\sum_{\substack{\{v^r\in\mathcal{V}^r\ :\\ v=\pi(v^r)\}}}x_{\mathcal{R}}(v^r,h,c^r),\ \ \forall v\in\mathcal{V},\forall h\in\Phi. \quad (17)$$

TABLE V
EVALUATION SETTINGS

Parameter	Values
Network topology	Abilene, Quest, OS3E, Bellcanada, Dfn
Ratio of multi-controller switches M_r	0, 0.25, 0.5, 0.75, 1
No. of vSDN requests $\|\mathcal{R}\|$	1, 3, 5, 7, 10, 15, 20, 40, 70, 100
No. of virtual nodes per vSDN $\|\mathcal{V}^r\|$	Uniformly distributed $2, \ldots, 10$
No. of virtual SDN controller c^r per request r	1
Virtual SDN controller placements (CP)	random (rnd), average (avg), maximum (max)
Virtual node locations $\pi(v^r)$	Uniformly selected from $v \in V$
HP objectives	max, maxavg, avg, avgmax
Runs per set-up	200

6) Multi-Controller Switches Constraint: We limit the number of special multi-controller SDN switches that are physically deployed in the network:

$$\sum_{v \in \mathcal{V}} x_{\mathcal{M}}(v) \leq M. \tag{18}$$

Note that via this constraint the four different architectures, as introduced in Section III, can be modeled, optimized, and analyzed. Setting $M = 0$ forces all $x_{\mathcal{M}}(v)$ to zero. Accordingly, there are no physical multi-controller SDN switches in the network, i.e., a physical SDN switch node can only be controlled by one hypervisor node. Thus, shared control domains, i.e., one node being controlled by multiple hypervisor nodes, are *not* possible.

VI. EVALUATION SET-UP

We extended our Python-based framework from [10] with our new models introduced in this paper. The framework uses Gurobi as solver for the MIP formulation. In this paper, the evaluation focuses mainly on the latency analysis, i.e., the hypervisor placement (HP) latency values (defined in Section V-B) and the latency values of the individual vSDN requests (defined in Section VII-B). Following [9], we use real network topologies to evaluate the architectures. The evaluation settings are summarized in Table V.

A. Substrate Networks

The performance evaluations focus initially on the hypervisor instance placement for the Internet2 Open Science, Scholarship and Services Exchange (OS3E) network topology. The OS3E network is a well known research network with OpenFlow capability. The OS3E network has 34 nodes and about 41 edges. The geographical node locations are used to calculate the latency of the network edges. We neglect additional latency, e.g., due to nodal processing. We conduct general topology evaluations for the Abilene, Quest, Bellcanada, OS3E, and Dfn networks [33].

B. Virtual SDN Network (vSDN) Request

For a given vSDN request r, the number of vSDN nodes, i.e., $|\mathcal{V}^r|$, is randomly determined by a uniform distribution between 2 and 10. The vSDN node locations are chosen randomly among all physical locations $\mathcal{V}$. The number of vSDN nodes per physical node is limited to one per request. The number of vSDN controllers per request r is set to one. In order to evaluate the impact of the virtual controller placement (CP) on the hypervisor placement (HP) latency, we consider three vSDN CPs, namely random (rnd), average (avg), and maximum (max). Random CP selects the node location $\pi(c^r)$ of the vSDN controller of a given request r randomly among all physical node locations $\mathcal{V}$. The average and maximum CPs [9] optimize the controller location for the locations of vSDN switches $\mathcal{V}^r$. The potential controller locations are always the set of physical node locations $\mathcal{V}$. For a given request $r \in \mathcal{R}$, the maximum CP minimizes the maximum control latency of all virtual switches $\mathcal{V}^r$ to their corresponding controller c^r. The average CP minimizes the average control latency for all controller c^r to switch connections per vSDN request. As we are interested in the study of a priori CPs, the vSDN controller locations are optimized a priori and fed as input into the MIP models.

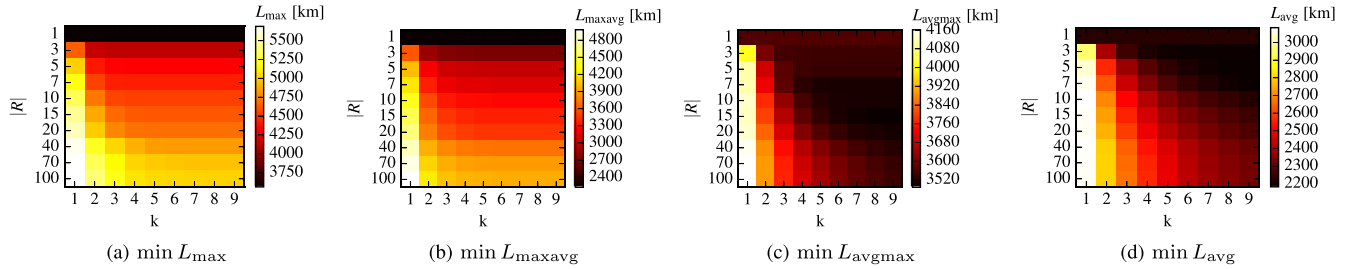
C. Architecture Comparison

For all architectures, we assume that all network nodes can host a hypervisor node, i.e., $\Phi = \mathcal{V}$. The number of hypervisor nodes k and the number of multi-controller switches M determine the type of hypervisor architecture. The centralized architecture, see Section III-A and Fig. 1a, is characterized by $k = 1$ and $M = 0$, i.e., each switch has only one controller (hypervisor) connection. Note also that $k > 0$ and $M = 0$ corresponds to the distributed architecture operating on single-controller switches (see Section III-B and Fig. 1b), while $0 < M < |\mathcal{V}|$ corresponds to the hybrid architecture (Section III-D, Fig. 1d) and $M = |\mathcal{V}|$ represents the distributed architecture where only multi-controller switches are deployed (see Section III-C, Fig. 1c). We set M through the ratio $M_r = M/|\mathcal{V}| = 0, 0.25, 0.5, 0.75, 1$ that specifies the maximum number of network nodes supporting the multi-controller feature. For instance, $M_r = 0.5$ corresponds to $M = 17$ multi-controller switches that can be placed inside the OS3E network. We initially compare all four SDN network hypervisor architectures in terms of the HP latency metrics defined in Section V-B. All latency results will be given in kilometers [km]. Subsequently, we analyze the latency values of the vSDN requests in order to evaluate the impact of virtualization. We then show through a generalized topology-aware analysis how the architectures behave for varying network topologies. Every optimization setup was executed 200 times to achieve statistically reliable results.

VII. EVALUATION RESULTS

A. Impact of Hypervisor Placement (HP) on Latency Metrics

We first present and discuss a compact representation of the results for varying number of vSDN requests $|\mathcal{R}|$ and increasing number of hypervisor instances k in Fig. 2. Based on our



(a) min $L_{\max}$ (b) min L_{maxavg} (c) min L_{avgmax} (d) min L_{avg}

Fig. 2. The heatmaps show the latency values (in kilometers [km]) averaged over 200 independent runs. Light yellow represents high latency values, while dark red represents low latency values. For each subfigure, the numbers of vSDN requests $|\mathcal{R}|$ are indicated on the left, the numbers of hypervisor instances k on the bottom, and the heatmap scale for the latencies on the right. Fixed param.: no multi-controller switches $M_r = 0$, random controller placement (CP).

observations we then conduct a more detailed evaluation of selected set-ups in Fig. 3 to clearly illustrate the effects of different architecture attributes, namely multi-controller switches, number of hypervisor instances k, and controller placements (CPs). In order to evaluate the virtualization overhead, i.e., the cost of virtualization, in terms of additional control plane latency, we conclude the OS3E evaluation by investigating the individual request latencies of the vSDN requests in Figs. 5–7. Finally, we provide an analysis of five different substrates in Figs. 8–10 to assess how our observations may be generalized.

1) Severe Impact of Number of vSDN Requests and Hypervisor Instances on HP Latency Metrics: Figures 2a–d provide a compact representation of the HP latency metrics for every combination of number of hypervisors k and number of vSDN requests $|\mathcal{R}|$. We consider the random CP strategy in order to focus on the impact of the parameters k and $|\mathcal{R}|$. The figures show heatmaps of the latency values averaged over 200 independent runs. The lowest latency value is represented in black color and the highest latency value in bright yellow color. Red represents intermediate latency values.

When only a single vSDN is considered ($|\mathcal{R}| = 1$), increasing the number of hypervisor instances k does not reduce any of the resulting latency metrics. When only a single hypervisor instance is considered ($k = 1$), the latencies significantly increase with increasing number of vSDN requests $|\mathcal{R}|$. On the other hand, for multiple requested vSDNs ($|\mathcal{R}| > 1$), we observe from Fig. 2 that increasing the number of hypervisor instances k generally reduces the latencies.

The number of requested vSDNs $|\mathcal{R}|$ plays an important role when optimizing the HP. For small $|\mathcal{R}|$, a small number of hypervisor instances k suffices to achieve optimal placements. In order to investigate the impact of k, M (M_r), and the CP in more detail, we set $|\mathcal{R}| = 70$ for the subsequent evaluations as this setting has shown a clear effect of increasing k on the HP latencies.

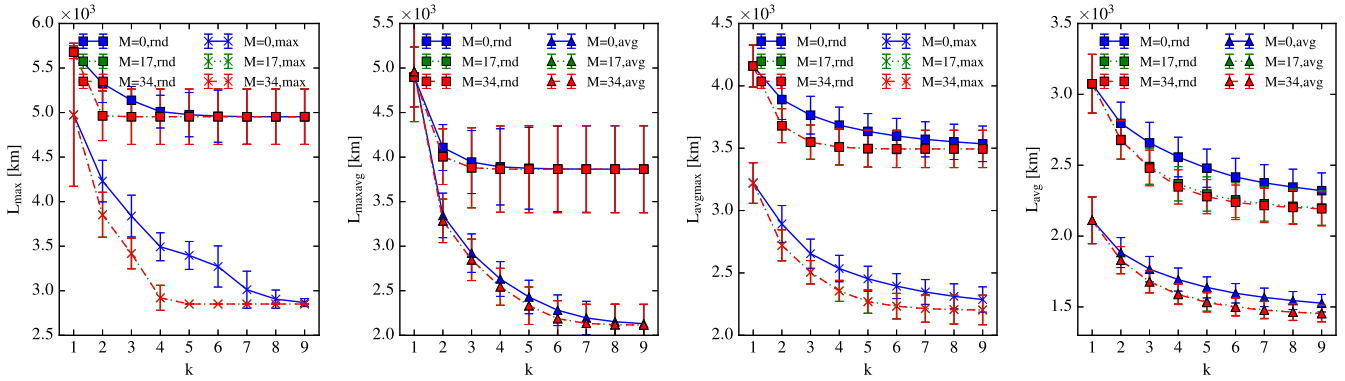
2) Increasing the Number of Hypervisor Instances k Minimizes Latency Metrics Differently: Figures 3a–d show the impact of the number of hypervisors k, the number of multi-controller switches M, and the virtual CPs on the achieved latencies. Each figure shows the result of one HP objective. Further, the random CP is compared to the best CP, i.e., either average or maximum CP, which achieved the best results in the conducted simulations. For each metric, the 95% confidence interval of the mean value over 200 runs is shown.

We observe from Figs. 3a–d that additional hypervisor instances generally reduce the latency objectives for all set-ups. This decrease of latencies with increasing k is consistent with the observations from Figs. 2a–d, which considered increasing k for a range of numbers of vSDN requests $|\mathcal{R}|$ (and $M_r = 0$). Notice in particular, the continuous drop of L_{avg} in Fig. 2d.

However, we also observe from Figs. 3a–d that for increasing k there is typically a point of diminishing returns, where adding hypervisor instances does not further reduce the latency. This point of diminishing returns varies according to latency objective and CP. For instance, the point of diminishing returns ranges from $k = 2$ for random CP with the $L_{\max}$ objective and $M = 34$ (Fig. 3a), to $k = 9$ for L_{avg} (Fig. 3d). That is, the convergence point differs strongly among the set-ups. Thus, in case of changing the operation goal of a hypervisor deployment, e.g., for $M_r = 0$ from L_{maxavg} to L_{avgmax}, a re-optimization of the HP may be necessary as a different number k of hypervisors may be needed for achieving an optimal latency value (e.g., from $k = 5$ for L_{maxavg} to $k = 9$ for L_{avgmax} with random CP).

3) More Multi-Controller Switches Demand Less Hypervisor Instances for an Optimal Solution: Figs. 3a–d also show that all objectives benefit from multi-controller switches. This means that increasing the number of multi-controller switches M decreases the number of hypervisor instances k required for an optimal solution. Further, the point of diminishing returns is affected. For instance, for $L_{\max}$ with random CP (Fig. 3a), $k = 2$ hypervisor instances achieve the lowest latency when $M = 17$ or 34, instead of $k = 5$ for $M = 0$. L_{avg} shows a more significant benefit of multi-controller switches over all k (Fig. 3d). This is shown by the non-overlapping blue solid ($M = 0$) and red dashed ($M = 34$) lines. To conclude, with respect to all objectives, only 50 % of switches need to support the multi-controller feature in order to achieve an optimal HP, as it is shown by the overlapping green dotted ($M = 17$) and red dashed ($M = 34$) lines.

4) The Best Controller Placement Strategy Depends on the Hypervisor Latency Objective: Figs. 3a–d indicate that optimized CP significantly decreases the values of all latency metrics, in some cases by more than 50 %. For instance, for the objective $L_{\max}$, the latency is reduced by nearly 42 %



(a) min $L_{\max}$ (b) min L_{maxavg} (c) min L_{avgmax} (d) min L_{avg}

Fig. 3. Latency values (95 % confidence intervals over 200 runs, in kilometers [km]) obtained with the different latency minimization objectives $L_{\max}$, L_{maxavg}, L_{avgmax}, and L_{avg} as a function of number of hypervisor instances k. The number of multi-controller switches is $M = 0$ ($M_r = 0$, solid lines), $M = 17$ ($M_r = 0.5$, green dotted lines), and $M = 34$ ($M_r = 1$, red dashed lines). The controller placement (CP) strategies are random (square boxes), maximum (crosses), and average (triangles).

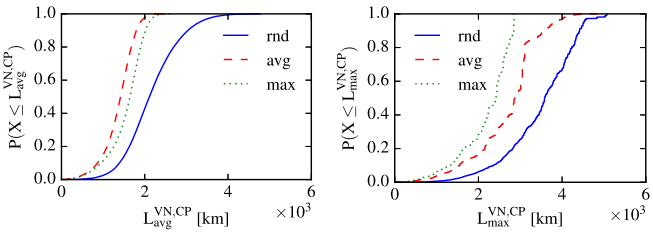
from an average value of $5 \cdot 10^3$ km to $2.9 \cdot 10^3$ km (Figs. 3a). The optimized CP also improves the centralized architecture ($k = 1$) for the $L_{\max}$, L_{avg}, and L_{avgmax} objectives. For L_{maxavg}, however, an optimized CP does not significantly reduce the latency of the centralized architecture ($k = 1$). Furthermore, the best CP strategy depends on the HP objective. The maximum CP achieves the most pronounced latency reduction for the $L_{\max}$ and L_{avgmax} latency objectives. For L_{avg} and L_{maxavg}, the average CP shows the best performance improvement.

5) The Average/Maximum Controller Placements Demand More Hypervisors for an Optimal Solution: In addition to reducing the latency values in general, the maximum and average CPs affect the point of diminishing returns with respect to the number of hypervisor instances k (Figs. 3a–d). Also, the number of multi-controller switches M impacts the convergence point per HP objective. For the L_{maxavg}, L_{avgmax}, and L_{avg} objectives, there is a small gap between $M = 0$ and $M = 34$. However, for $L_{\max}$, there is a pronounced gap between $M = 0$ and $M = 34$; and only for $k = 9$ hypervisor instances do the $M = 0$ and $M = 34$ curves converge. For the L_{maxavg} objective, the convergence point is also only reached for $k = 9$ hypervisor instances. When comparing all latency values for $k = 1$, only L_{maxavg} benefits neither from optimized CP nor from multi-controller switches. This effect can be explained by the examination of the individual latencies of the vSDN requests, as conducted in the next subsection.

B. Analysis of the vSDN Requests' Control Plane Latency—The Cost of Virtualization

Before analyzing the impact of the HP on the individual vSDN requests, we first examine the impact of the CP on the individual requests without virtualization. This means that we calculate for each request the best possible latency values, which are determined by the CP. Without virtualization, the connections between the requested switches and controllers do not have to pass through any hypervisor instance. We define the maximum request latency

$$L_{\max}^{VN,CP}(r) = \max_{v^r \in \mathcal{V}^r} d\big(\pi\left(v^r\right), \pi\left(c^r\right)\big), \quad \forall r \in \mathcal{R} \tag{19}$$



(a) Average request latency (b) Maximum request latency

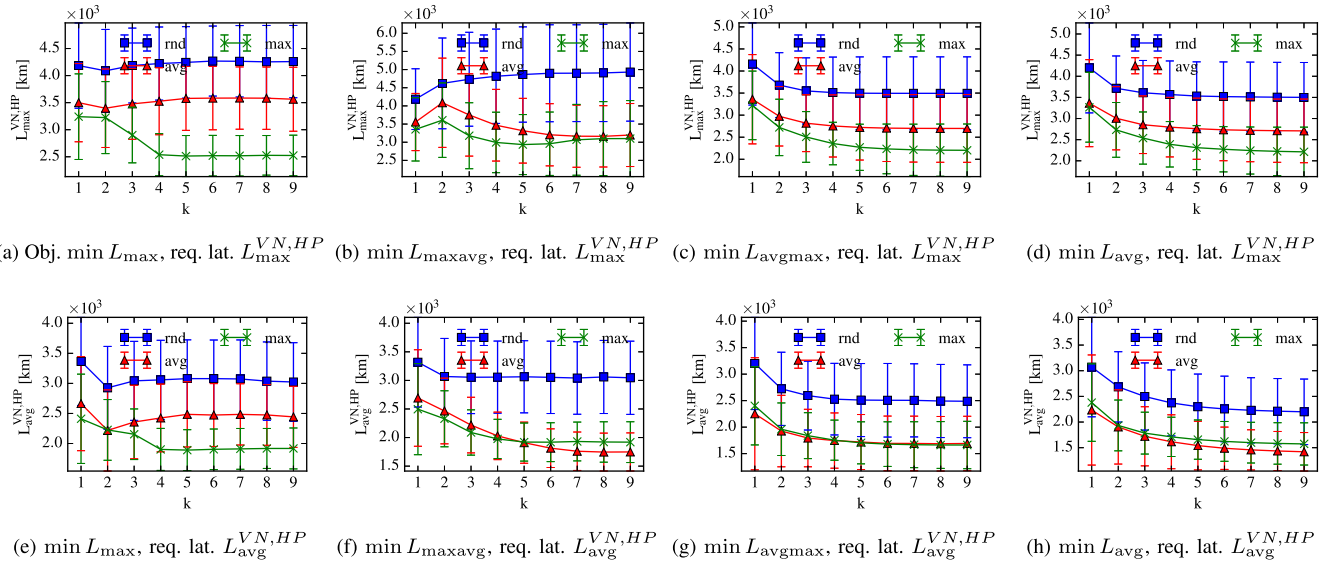
Fig. 4. Cumulative distribution functions of average ($P(X \leq L_{\text{avg}}^{VN,CP})$) and maximum ($P(X \leq L_{\max}^{VN,CP})$) latencies for direct virtual switch to controller connections of individual requested vSDNs $r \in \mathcal{R}$, without traversing hypervisors. The controller placement (CP) strategies are: random (blue solid line), average (green dotted line), and maximum (red dashed line).

and the average request latency

$$L_{\text{avg}}^{VN,CP}(r) = \frac{1}{|\mathcal{V}^r|} \sum_{v^r \in \mathcal{V}^r} d\big(\pi\left(v^r\right), \pi\left(c^r\right)\big), \quad \forall r \in \mathcal{R}. \tag{20}$$

Note that these are the definitions of the request latencies without any virtualization. For calculating the latencies with virtualization $L_{\text{avg}}^{VN,HP}(r)$ and $L_{\max}^{VN,HP}(r)$, $d(\pi(v^r), \pi(c^r))$ needs to be replaced by $d(\pi(v^r), h, \pi(c^r))$, i.e., by the distance of the paths via the used hypervisor instances. We omit the request specification '(r)' in the following to avoid notational clutter.

Figs. 4a–b show the $L_{\text{avg}}^{VN,CP}$ and $L_{\max}^{VN,CP}$ CDFs for the random, average, and maximum CPs without virtualization (i.e., no HP). In general, they show the best possible request latencies that can be achieved for each request. Virtualization, i.e., hypervisor placement, will in the best case achieve the latency values as shown by the figures. The maximum and the average placement strategy reduce the request latency values $L_{\text{avg}}^{VN,CP}$ and $L_{\max}^{VN,CP}$. The average CP achieves the lowest latency values for $L_{\text{avg}}^{VN,CP}$, while the maximum CP achieves the lowest latencies for $L_{\max}^{VN,CP}$. Interestingly, the results of the maximum CP are close to the average CP for $L_{\text{avg}}^{VN,CP}$. The reason is that the maximum CP places the controller in the middle of the longest path between two virtual SDN switches to reduce $L_{\max}^{VN,CP}$. This



(a) Obj. min $L_{\max}$, req. lat. $L_{\max}^{VN,HP}$ (b) min L_{maxavg}, req. lat. $L_{\max}^{VN,HP}$ (c) min L_{avgmax}, req. lat. $L_{\max}^{VN,HP}$ (d) min L_{avg}, req. lat. $L_{\max}^{VN,HP}$

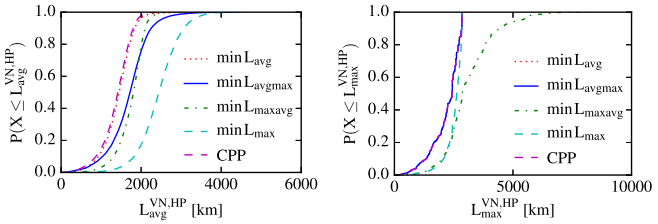
(e) min $L_{\max}$, req. lat. $L_{\text{avg}}^{VN,HP}$ (f) min L_{maxavg}, req. lat. $L_{\text{avg}}^{VN,HP}$ (g) min L_{avgmax}, req. lat. $L_{\text{avg}}^{VN,HP}$ (h) min L_{avg}, req. lat. $L_{\text{avg}}^{VN,HP}$

Fig. 5. Mean values and 95 % confidence intervals of average ($L_{\max}^{VN,HP}$) and maximum ($L_{\text{avg}}^{VN,HP}$) latencies for virtual switch-hypervisor-controller connections of individual vSDNs $r \in \mathcal{R}$. For each HP latency minimization objective, the impact of k hypervisor instances and the controller placement (CP) are depicted: random CP (blue boxes), average CP (red triangles), and maximum CP (green crosses). Fixed param.: $M_r = 0.5$ multi-contr. switches.

is in most cases a central position of the vSDN, which leads also to low $L_{\text{avg}}^{VN,CP}$ values.

Figures 5a–h show the impact of CPs and the number of hypervisor instances k on the request latencies $L_{\max}^{VN,HP}$ and $L_{\text{avg}}^{VN,HP}$. Each figure shows the behavior for a given HP objective. For distributed architectures ($k > 1$), we set the number of multi-controller switches to $M = 17$ as the hybrid architecture has already optimal HP latency values.

1) Adding Hypervisor Instances May Increase the Request Latency With Maximum-Based Objectives: For the maximum-based latency objectives, namely $L_{\max}$, which considers the maximum of all individual path latencies $d(\pi(v^r), h, \pi(c^r))$, $v^r \in \mathcal{V}^r$ of all requests $r \in \mathcal{R}$ (see Eqn. (1)), and L_{maxavg}, which considers the maximum of the average vSDN (request) latencies (see Eqn. (9)), we observe from Figs. 3a, b, e, and f mixed behaviors. For instance, for the maximum CP, which achieves generally the lowest individual maximum request latencies $L_{\max}^{VN,HP}$, additional hypervisor instances are beneficial for the $L_{\max}$ objective, but may increase latencies for the L_{maxavg} objective. Similarly, additional hypervisors increase the request latencies for several other combinations of CP and request latency metric in Figs. 3a, b, e, and f. This is because the maximum-based latency objectives $L_{\max}$ strive to minimize the maximum path latency over all requested vSDNs (see Eqn. (1)). For this, $L_{\max}$ relaxes the maximum request latency $L_{\max}^{VN,HP}(r)$ and average request latency $L_{\text{avg}}^{VN,HP}(r)$ for some vSDN requests r in order to improve the maximum latency over all requests. Similarly, L_{maxavg} strives to minimize the maximum average request latency $L_{\text{avg}}^{VN,HP}(r)$ over all requested vSDNs (see Eqn. (9)). Thus, a single vSDN request, namely the vSDN with the longest virtual node-hypervisor-controller path (for $L_{\max}$) or the highest average request latency (for L_{maxavg}) governs the optimal latency objective value. For the remaining vSDN requests, i.e., the requests that do not affect the objective,

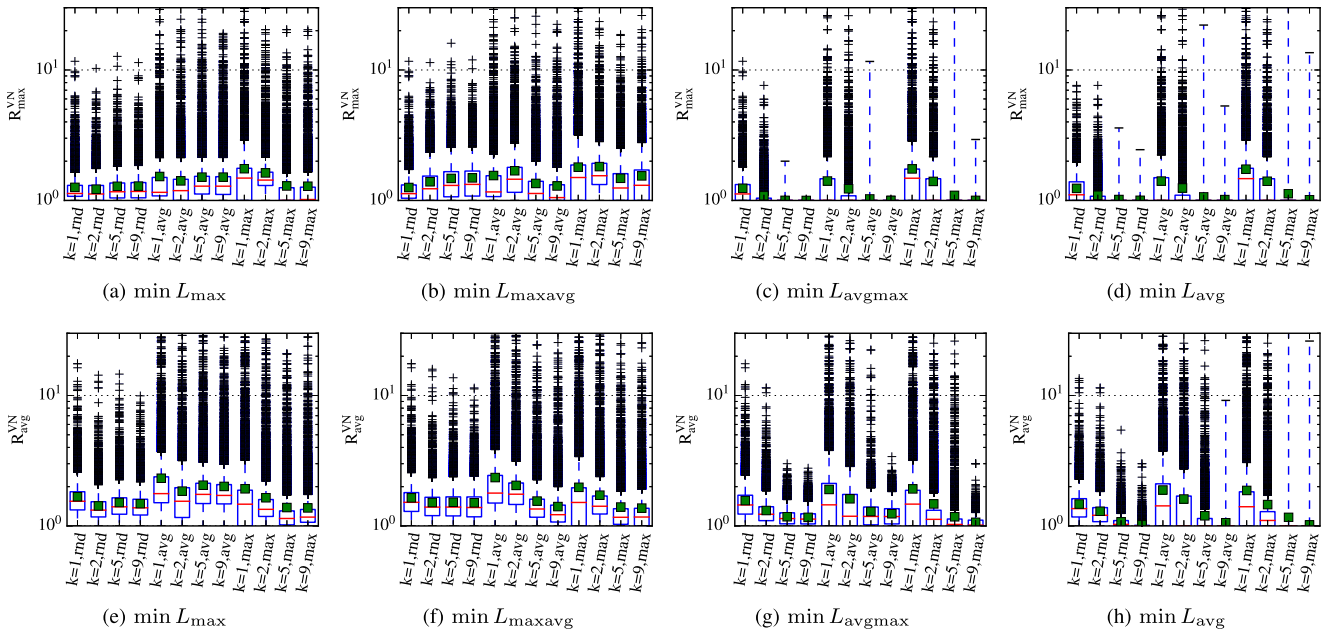


(a) Average request latency (HP) (b) Maximum request latency (HP)

Fig. 6. Cumulative distribution functions of average ($P(X < L_{\text{avg}}^{VN,HP})$) and maximum ($P(X < L_{\max}^{VN,HP})$) individual vSDN request latencies with HP (virtualization); $L_{\max}^{VN,CP}$ and $L_{\text{avg}}^{VN,CP}$ show the request latencies without virtualization (see Fig. 4). Fixed param.: $k = 9$ hypervisors, $M_r = 0.5$ multi-contr. switches.

the responsible hypervisors may not be placed optimally with respect to $L_{\max}^{VN,HP}$ and $L_{\text{avg}}^{VN,HP}$. Therefore, some vSDN requests may experience increased latencies when adding hypervisors in order to improve the optimal latency objective value, as demonstrated by Figs. 3a, b, e, and f. We plan to propose an algorithm that addresses this issue in future work.

2) Average-Based Latency Objectives Always Benefit From Additional Hypervisor Instances: We observe from Fig. 5c, d, g, and h that for the average-based latency objectives L_{avgmax} and L_{avg}, the individual requests always benefit from additional hypervisor instances, i.e., from increasing k. Through the averaging over all path lengths (L_{avg}) or the maximum path lengths of all vSDN requests (L_{avgmax}), the average-based latency metrics consider all vSDN requests and exploit additional hypervisor instances to achieve lower latency objectives and lower individual vSDN request latencies. We also observe from Figs. 5c, d that the maximum CP achieves the lowest maximum request latencies $L_{\max}^{VN,HP}$ while the average CP achieves the lowest average request latencies $L_{\text{avg}}^{VN,HP}$ (Figs. 5g, h). Overall, the objective L_{avg} (Figs. 5d, h) achieves the lowest request latencies $L_{\max}^{VN,HP}$ and $L_{\text{avg}}^{VN,HP}$.



(a) min $L_{\max}$ (b) min L_{maxavg} (c) min L_{avgmax} (d) min L_{avg}

(e) min $L_{\max}$ (f) min L_{maxavg} (g) min L_{avgmax} (h) min L_{avg}

Fig. 7. Boxplots for the maximum and average latency overhead ratios $R^{VN}_{\max}$ and R^{VN}_{avg} (Eqs. (21) and (22)) for OS3E network. An overhead ratio of one corresponds to *no* overhead, i.e., a *zero cost of virtualization.* The blue boxes show the upper 75 % quartile and the lower 25 % quartile. The green filled squares show the mean and the red line the median. In case the upper quartile and the lower quartile are equal, the whiskers reach the maximum outlier value, shown via blue dashed lines. The black crosses indicate the outliers that do not fall into the 1.5 times interquartile range of the whiskers. For each figure, $k = 1, 2, 5, 9$ hypervisor instances are compared for the controller placement (CP) strategies (rnd, max, avg). **Y-axes are scaled logarithmically.**

3) Significant Request Latency Trade-Offs Among All Objectives Can be Observed: In order to achieve their optimization goal, the objectives lead to trade-offs among the request latencies $L^{VN,HP}_{\max}$ and $L^{VN,HP}_{\text{avg}}$. We illustrate these trade-offs for the hybrid architecture ($M = 17$) with $k = 9$ hypervisor instances. The following observations hold in general also for most other set-ups. As depicted in Fig. 6a, the L_{avg} objective achieves the lowest request latencies. We observe a clear trade-off between the L_{avgmax} and L_{maxavg} objectives with respect to $L^{VN,HP}_{\text{avg}}$. As expected, L_{maxavg} pushes down the maximum average latency among all requests, thus, achieving lower latencies for the upper 20 % of the requests. By pushing down the individual maximum path latencies over all requests, L_{avgmax} pays more attention to the individual paths, i.e., controller to switch connections, of the requests. Consequently, L_{avgmax} accepts larger values for 20 % of the requests in order to improve the latency of the 80 % remaining requests.

Fig. 6b shows again important trade-offs among all objectives. Although $L_{\max}$ minimizes the maximum request latency, it accepts overall worse request latencies than L_{avg} and L_{avgmax}. Further, the min L_{maxavg} curve illustrates the model's working behavior when optimizing for L_{maxavg}. While minimizing L_{maxavg} pushes the maximum average latencies of all requests down (Fig. 6a), it relaxes the request latencies $L^{VN,HP}_{\max}$ towards higher values (Fig. 6b).

4) Controller Placement Strategy and Additional Hypervisor Instances Can Significantly Reduce Virtualization Overhead: Having observed that the different latency objectives provide generally varying trade-offs between the request latencies, we now analyze the virtualization overhead per vSDN request in detail. In order to investigate how much overhead virtualization adds to the request latency, we introduce metrics that reflect the virtualization overhead ratio, i.e., the cost of virtualization. We define the maximum latency overhead ratio

$$R^{VN}_{\max}(r) = \frac{L^{VN,HP}_{\max}(r)}{L^{VN,CP}_{\max}(r)}, \ \forall r \in \mathcal{R} \tag{21}$$

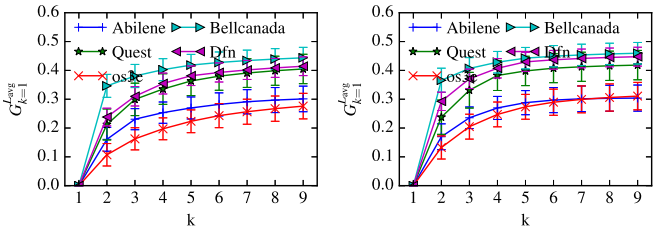
and the average latency overhead ratio

$$R^{VN}_{\text{avg}}(r) = \frac{L^{VN,HP}_{\text{avg}}(r)}{L^{VN,CP}_{\text{avg}}(r)}, \ \forall r \in \mathcal{R}. \tag{22}$$

A request is affected by virtualization if an overhead ratio is larger than one. An overhead ratio of one means that the request latency is not increased by virtualization.

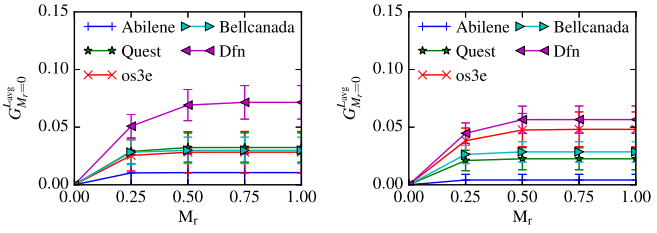
For analysis, the distributed hybrid architecture ($k > 1, M = 17$) is chosen as it has shown an optimal performance for the HP latency objectives. We selected $k = 1, 2, 5, 9$ to provide a representative set to illustrate the impact of using additional hypervisor instances. Figs. 7a–h represent the latency overhead ratios of all latency objectives. Boxplots depict how additional hypervisor instances and the CP impact the overhead ratios. As shown by Fig. 7, for some vSDN requests the controller latency is up to 15 times higher. The random CP has the lowest virtualization overhead. This is because the random CP has already relative high latencies $L^{VN,CP}_{\text{avg}}$ and $L^{VN,CP}_{\max}$, see Fig. 4.

Generally, we observe from Fig. 7 that the objectives L_{avgmax} and L_{avg} achieve the lowest overheads. Specifically, for $R^{VN}_{\max}$, the objectives L_{avgmax} and L_{avg} achieve decreasing latency overheads as more hypervisor instances are deployed, i.e., k is increased. More than 75 % of the requests (Fig. 7c and Fig. 7d) achieve an overhead ratio



(a) $M_r = 0$ (b) $M_r = 1$

Fig. 8. Latency reduction due to adding hypervisor instances for different substrate topologies (indicated by line styles and colors). Multi-controller ratios $M_r = 0$ and $M_r = 1$ are compared for average CP.



(a) $k = 2$, average CP (b) $k = 9$, average CP

Fig. 9. Relative latency reduction due to increasing ratio M_r of multi-controller switches in 0.25 steps for different topologies (indicated by line styles and colors). Distributed architectures are compared for $k = 2$ and 9 hypervisor instances for average CP.

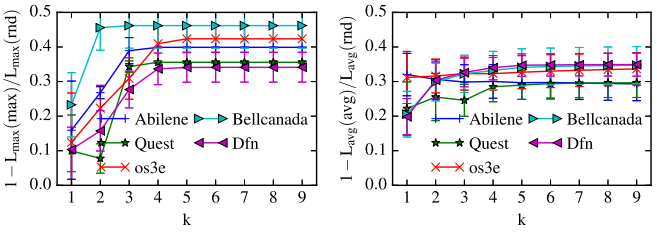
$R^{VN}_{\max} = 1$, i.e., their maximum latencies are not increased at all by virtualization, when $k = 5$ or 9. In contrast, the maximum-based latency objectives $L_{\max}$ and $L_{\max\text{avg}}$ exhibit again the mixed behavior for increasing k as observed in Section VII-B1.

For R^{VN}_{avg}, the objectives $L_{\max\text{avg}}$, L_{avgmax}, and L_{avg} benefit from additional hypervisors for all CP strategies. To conclude, with a moderately high number of hypervisor instances ($k = 5$ or 9), the average-based latency objectives L_{avgmax} and L_{avg} have demonstrated the lowest overhead ratios, irrespective of the CP strategy. Thus, when individual request latencies need to be optimized, the objectives L_{avg} and L_{avgmax} should be chosen over $L_{\max}$ or $L_{\max\text{avg}}$.

C. Analysis of Different Substrate Network Topologies

We now examine the impact of different network topologies. The goal of this examination is to determine whether some of the observations and conclusions from the OS3E network can be generalized to other network topologies. We focus on the L_{avg} HP latency minimization objective as it has generally achieved low latency values so far, including for the individual request latencies $L^{VN,HP}_{\max}$ and $L^{VN,HP}_{\text{avg}}$. The substrate topologies have varying numbers of network nodes and links. We set the number of requested vSDNs to $|\mathcal{R}| = 70$ to allow for a close comparison to the preceding detailed analysis. Throughout, we present the results as relative values, i.e., the performance gain of a specific feature is compared to a baseline set-up, in order to facilitate comparisons across different network topologies.

1) Impact of Adding Hypervisor Instances: We start to examine the impact of adding hypervisor instances, i.e., we evaluate the latency reduction (performance gain) $G^{L_{\text{avg}}}_{k=1} = 1 - L_{\text{avg}}(k = x)/L_{\text{avg}}(k = 1)$. $L_{\text{avg}}(k = x)$ denotes the HP latency for x hypervisor instances and $L_{\text{avg}}(k = 1)$ is the latency of the centralized architecture. A higher ratio indicates a better objective improvement (latency reduction). Figs. 8a–b show the ratios when using the average CP for up to $k = 9$ hypervisor instances. The latency reduction can reach 40 %, even without ($M_r = 0$) multi-controller switches (Fig. 8a). As already seen for the OS3E topology, the improvement slowly converges from $k = 5$ onward. This also holds for the distributed architectures, where all switches ($M_r = 1$) can operate in multi-controller mode (Fig. 8b).



(a) $L_{\max}$, maximum CP (b) L_{avg}, average CP

Fig. 10. Latency reduction due to maximum and average CP relative to random CP for different topologies (indicated by line styles and colors) for $k = 1, \ldots, 9$ hypervisor instances for $M_r = 1$.

2) Impact of Adding Multi-Controller Switches: We proceed to examine the performance gain from adding multi-controller switches. We evaluate the relative performance gain (latency reduction) $G^{L_{\text{avg}}}_{M_r=0} = 1 - L_{\text{avg}}(M_r = x)/L_{\text{avg}}(M_r = 0)$ when increasing the ratio (proportion) of multi-controller switches from $M_r = 0$ to $M_r = x = 0.25, 0.5, 0.75, 1$. We focus on $k = 2$ and 9 hypervisor instances. When $M_r = 0.5$ (50 %) multi-controller switches are deployed, an architecture with $k = 2$ hypervisor instances can achieve up to 8 % performance gain (Fig. 9a). Generally, larger topologies (Dfn) benefit more from the multi-controller feature than smaller topologies (Abilene). The point of diminishing returns of the considered topologies ranges from $M_r = 0.25$ to 0.5. For $k = 9$ hypervisor instances, the performance gain is slightly lower than for $k = 2$ instances. Again, larger topologies, such as Dfn, benefit more from the deployment of multi-controller switches than smaller topologies.

3) Impact of Controller Placement CP Strategies: Finally, we investigate the performance gains due to CP strategies. We evaluate the relative performance gain of the maximum and average CP versus the random CP. We allow all network nodes to provide the multi-controller feature ($M_r = 1$). Figs. 10a–b show the performance gain for $L_{\max}$ and L_{avg} as a function of the number of hypervisor instances k. The maximum CP leads to performance gain over all topologies between 0.3 and 0.5. In both cases, the point of diminishing returns is $k = 4$, which would be the preferred number of hypervisor instances in those set-ups.

VIII. Conclusion

When virtualizing software-defined networks, the control plane latency plays an important role for the performance

of the individual virtual SDN networks (vSDNs). In particular, when providing programmability and virtualization in future communication networks, such as Internet of Things and 5G networks [34], [35], low control plane latencies are important. In this article, we have investigated the hypervisor placement, i.e., the placement of the hypervisor instances that provide the virtualization functionality. We have defined mixed integer programming models for a centralized and three distributed SDN network virtualization hypervisor architectures. Furthermore, we have investigated the impact of multi-controller switches that can simultaneously connect to multiple hypervisor instances. For evaluation of the four modeled architectures, we have investigated the impact of the hypervisor placement on the control plane latencies of the entire network as well as individual vSDNs. We have identified the control plane latency overhead due to the requirement that the SDN switch to controller connections traverse a hypervisor instance for virtualization. This latency overhead represents the cost of virtualization. We have observed that virtualization can add significant control latency overhead for individual vSDNs. However, we have also shown that adding hypervisor instances and using flexible multi-controller switches can significantly reduce the hypervisor latencies for a range of different substrate network topologies. Overall, the introduced optimization models provide network operators with a formal mechanism to rigorously examine the trade-offs of SDN hypervisor placement and multi-controller SDN switch usage for vSDNs.

Important directions for future research include the extension of the hypervisor placement study to a wider set of performance metrics. For instance, to reduce energy consumption, vSDN assignments can be consolidated on hypervisor instances at runtime. As such consolidations would modify established control plane paths, thorough planning and optimization of such consolidation operations are needed to avoid control plane interruptions. Moreover, a high number of assigned vSDNs per hypervisor may overload the hypervisor CPU. Thus, load balancing schemes may need to balance the number of physical switches, virtual switches, and tenant controllers that are assigned to a given hypervisor. While this study found that multi-controller switches reduce the hypervisor control plane latency, their use for reliability or load balancing has not yet been investigated, presenting important future work directions.

When extending the network model from Section IV-A to limited link (edge) capacities, a packet-based optimization may become necessary. For instance, packet-level congestion problems may need to be addressed, e.g., through traffic shaping. As network virtualization allows for flexible dynamic adaptation of virtual networks at runtime, runtime updates of hypervisor instances are another important research direction. More specifically, new hypervisor placement models should be developed to dynamically plan and optimize the hypervisor placements as the virtual network demands fluctuate over time. Such optimization models might require considering the different migration and state synchronization techniques that are needed when adapting placements at runtime.

REFERENCES

[1] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart. 2016.

[2] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[3] R. Sherwood *et al.*, "Carving research slices out of your production networks with OpenFlow," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 129–130, Jan. 2010.

[4] A. Al-Shabibi *et al.*, "OpenVirteX: A network hypervisor," in *Proc. Open Netw. Summit*, Santa Clara, CA, USA, Mar. 2014, pp. 1–2.

[5] A. Blenk, A. Basta, and W. Kellerer, "HyperFlex: An SDN virtualization architecture with flexible hypervisor function allocation," in *Proc. IFIP/IEEE IM*, Ottawa, ON, Canada, May 2015, pp. 397–405.

[6] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 655–685, 1st Quart. 2016.

[7] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proc. ACM Workshop HotSDN*, Chicago, IL, USA, 2014, pp. 1–6.

[8] OpenDaylight. (2013). *A Linux Foundation Collaborative Project*. [Online]. Available: http://www.opendaylight.org

[9] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 473–478, Sep. 2012.

[10] A. Blenk, A. Basta, J. Zerwas, and W. Kellerer, "Pairing SDN with network virtualization: The network hypervisor placement problem," in *Proc. IEEE NFV-SDN*, San Francisco, CA, USA, 2015, pp. 198–204.

[11] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[12] ONF. (Dec. 2014). *OpenFlow Switch Specifications 1.5*. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.5.pdf

[13] Hewlett-Packard. (Jun. 2015). *HP Switch Software OpenFlow v1.3 Administration Guide K/KA/WB 15.17*. [Online]. Available: http://h10032.www1.hp.com/ctg/Manual/c04656675

[14] K. Aardal, M. Labbé, J. Leung, and M. Queyranne, "On the two-level uncapacitated facility location problem," *INFORMS J. Comput.*, vol. 8, no. 3, pp. 289–301, Aug. 1996.

[15] A. Klose and A. Drexl, "Facility location models for distribution system design," *Eur. J. Oper. Res.*, vol. 162, no. 1, pp. 4–29, 2005.

[16] H. Pirkul and V. Jayaraman, "A multi-commodity, multi-plant, capacitated facility location problem: Formulation and efficient heuristic solution," *Comput. Oper. Res.*, vol. 25, no. 10, pp. 869–878, 1998.

[17] S. Guha, A. Meyerson, and K. Munagala, "Hierarchical placement and network design problems," in *Proc. IEEE FOCS*, Redondo Beach, CA, USA, 2000, pp. 603–612.

[18] R. Z. Farahani, M. Hekmatfar, B. Fahimnia, and N. Kazemzadeh, "Hierarchical facility location problem: Models, classifications, techniques, and applications," *Comput. Ind. Eng.*, vol. 68, no. 1, pp. 104–117, 2014.

[19] S. Lange *et al.*, "Heuristic approaches to the controller placement problem in large scale SDN networks," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 1, pp. 4–17, Mar. 2015.

[20] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 18, no. 8, pp. 1339–1342, Aug. 2014.

[21] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 19, no. 1, pp. 30–33, Jan. 2015.

[22] Y. Jiménez, C. Cervelló-Pastor, and A. J. García, "On the controller placement for designing a distributed SDN control layer," in *Proc. IFIP Netw.*, Trondheim, Norway, Jun. 2014, pp. 1–9.

[23] S. Lange *et al.*, "Specialized heuristics for the controller placement problem in large scale SDN networks," in *Proc. ITC*, Ghent, Belgium, Sep. 2015, pp. 210–218.

[24] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "On reliability-optimized controller placement for software-defined networks," *China Commun.*, vol. 11, no. 2, pp. 38–54, Feb. 2014.

[25] L. F. Müller, R. R. Oliveira, M. C. Luizelli, L. P. Gaspary, and M. P. Barcellos, "Survivor: An enhanced controller placement strategy for improving SDN survivability," in *Proc. IEEE GLOBECOM*, Austin, TX, USA, Dec. 2014, pp. 1909–1915.

[26] M. F. Bari *et al.*, "Dynamic controller provisioning in software defined networks," in *Proc. CNSM*, Zürich, Switzerland, Oct. 2013, pp. 18–25.

[27] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, 4th Quart. 2013.
[28] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, Apr. 2008.
[29] M. Demirci and M. Ammar, "Design and analysis of techniques for mapping virtual networks to software-defined network substrates," *Comput. Commun.*, vol. 45, pp. 1–10, Jun. 2014.
[30] M. Chowdhury, M. R. Rahman, and R. Boutaba, "ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 206–219, Feb. 2012.
[31] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Comput.*, vol. 17, no. 2, pp. 20–27, Mar./Apr. 2013.
[32] A. Basta, A. Blenk, H. B. Hassine, and W. Kellerer, "Towards a dynamic SDN virtualization layer: Control path migration protocol," in *Proc. CNSM*, Barcelona, Spain, Nov. 2015, pp. 354–359.
[33] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.
[34] N. Omnes, M. Bouillon, G. Fromentoux, and O. Le Grand, "A programmable and virtualized network & IT infrastructure for the Internet of Things: How can NFV & SDN help for facing the upcoming challenges," in *Proc. IEEE ICIN*, Paris, France, Feb. 2015, pp. 64–69.
[35] E. Hossain and M. Hasan, "5G cellular: Key enabling technologies and research challenges," *IEEE Instrum. Meas. Mag.*, vol. 18, no. 3, pp. 11–21, Jun. 2015.

Andreas Blenk received the Diploma degree in computer science from the University of Würzburg, Germany, in 2012. He is currently pursuing the Ph.D. degree with the Technische Universität München (TUM). In 2012, he joined the Chair of Communication Networks with the TUM, where he is a Research and Teaching Associate, and also a member of the Software Defined Networking and Network Virtualization Research Group. His research is focused on service-aware network virtualization, virtualizing software defined networks, as well as resource management and embedding algorithms for virtualized networks.

Arsany Basta received the M.Sc. degree in communication engineering from the Technische Universität München (TUM), in 2012, where he is currently pursuing the Ph.D. degree. He joined the TUM Institute of Communication Networks, in 2012, as a member of the research and teaching staff. His current research focuses on the applications of software defined networking, network virtualization, and network function virtualization to the mobile core toward the next generation (5G) network.

Johannes Zerwas (S'14) received the B.Sc. degree in electrical engineering and information technology from the Technische Universität München, Munich, Germany, in 2015, where he is currently pursuing the M.Sc. degree, and is also a Research Assistant with the Chair of Communication Networks.

Martin Reisslein (S'96–A'97–M'98–SM'03–F'14) received the Ph.D. in systems engineering from the University of Pennsylvania, in 1998. He is a Professor with the School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe. He served as an Editor-in-Chief for the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, from 2003 to 2007, and as an Associate Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING, from 2009 to 2013. He currently serves as an Associate Editor for the IEEE TRANSACTIONS ON EDUCATION, *Computer Networks*, and *Optical Switching and Networking*.

Wolfgang Kellerer (M'96–SM'11) has been a Full Professor with the Technische Universität München, heading the Chair of Communication Networks with the Department of Electrical and Computer Engineering, since 2012. He was the Director and the Head of Wireless Technology and Mobile Network Research with NTT DOCOMO's European Research Laboratories and DOCOMO Euro-Laboratory, for over ten years. His research focuses on concepts for the dynamic control of networks (software defined networking), network virtualization and network function virtualization, application-aware traffic management, and machine-to-machine communication, device-to-device communication, and wireless sensor networks with a focus on resource management toward a concept for 5th generation mobile communications in wireless networks. His research resulted over 200 publications and 29 granted patents in the areas of mobile networking and service platforms. He is a member of ACM and the VDE ITG.