Delay-Aware Dynamic Hypervisor Placement and Reconfiguration in Virtualized SDN

Saquib Amjad, Amir Varasteh, Nemanja Deric, Carmen Mas-Machuca
Chair of Communication Networks, Department of Electrical and Computer Engineering,
Technical University of Munich, Germany
Email:{saquib.amjad, amir.varasteh, nemanja.deric, cmas}@tum.de

***Abstract*—Software defined networking (SDN) provides different functionality and resource sharing capabilities with the aid of virtualization. In virtualized SDN, multiple SDN tenants can bring their controllers and different functions in the same physical substrate. The SDN hypervisor provides the link between the physical network substrate and its SDN tenants. Distributed hypervisor architecture can handle scalability, virtualization, and reassignment better than a centralized hypervisor architecture. However, distributed hypervisors may require synchronization and load balancing. Due to the dynamic control plane traffic, some network elements may suffer from increase in delay. Controller placement problem (CPP) and hypervisor placement problem (HPP) have tackled the static placement and mapping strategies.**

In this paper, we design a dynamic hypervisor assignment and reconfiguration with load balancing using integer linear programming. The proposed model focuses on optimizing multiple objectives: control plane latency, processing latency, and load balancing for distributed hypervisors. We provide different heuristics for the model and then perform evaluation on two topologies- Abilene and EU-nobel. The results show the trade off between load balancing and minimizing latency, and comparison of the performance of heuristics and MILP model.

***Index Terms*—virtual network, network virtualization, hypervisor migration, dynamic hypervisor placement, load balancing, network reconfiguration**

I. Introduction

In the last decade, software-defined networking (SDN) has emerged as a versatile paradigm of programmable and configurable networks with decoupled control and data plane. SDN, combined with Network Virtualization (NV), presents a scalable, programmable network with advanced resource sharing and abstraction. The tenant networks, called virtual SDNs (vSDNs), can employ their controllers [1], respective policies, and program their resources independently [2]. Virtualization enables network planners to increase network resource utilization by adding multiple network slices on the same physical substrate network. Network virtualization is achieved by using an SDN network virtualization hypervisor layer between the SDN controller and its tenant switches. The hypervisor provides network abstraction transparently to its vSDN controllers. In this work, we consider distributed hypervisor architecture. A centralized hypervisor cannot provide the same level of scalability and efficient virtualization as distributed hypervisors. The control plane latency is an important network performance measure, where high latency leads to packet losses and long flow setup times. However, virtualization adds overhead in control plane latency due to the introduction of the hypervisor layer. Furthermore, the current end-user demands include resource-hungry services such as video streaming, voice over IP, etc., which require stringent QoS standards at low latencies. As a network operator, planning a network with low latencies is a constant challenge.

978-1-6654-2434-9/21/$31.00 ©2021 IEEE

In this paper, we model and analyze a hypervisor placement strategy that can adapt to the dynamic demands in the network by vSDN to hypervisor reassignments, while providing low latencies for all its tenant networks. With the increasing use of distributed hypervisor architecture [3] and hypervisor migration protocols [4], the proposed model takes this dynamicity into account and allows network tenants to choose hypervisor instances based on their latencies, while also allowing hypervisor instances to allocate vSDNs dynamically to prevent capacity overload. To focus on the effect of hypervisor placement and vSDN reassignment, we fix the controller locations for all time. This model can answer the following issues:

- Number of hypervisors required in the network and their locations.
- Effect of control plane traffic on the hypervisor to switch mapping.
- Dynamic load distribution based on current hypervisor load and remaining processing capacity.

The rest of the paper is structured as follows. Section II motivates the topic with an example scenario. Section III highlights the related works in controller and hypervisor placement problems, specifically focusing on dynamic placement and existing reassignment and migration models and protocols. The system model and MILP formulation are presented in Section IV. Section V introduces the proposed heuristics for placement and reassignment sub-problems. The performance evaluation is conducted in Section VI, followed by the conclusion of the work and possible future directions in Section VII.

II. Motivation

The SDN network performance is dependent on the locations of its network entities, such as controllers and hypervisors, and the effects of dynamic network traffic. In a non virtualized SDN environment, the controller placement problem (CPP) was introduced to find the optimal number

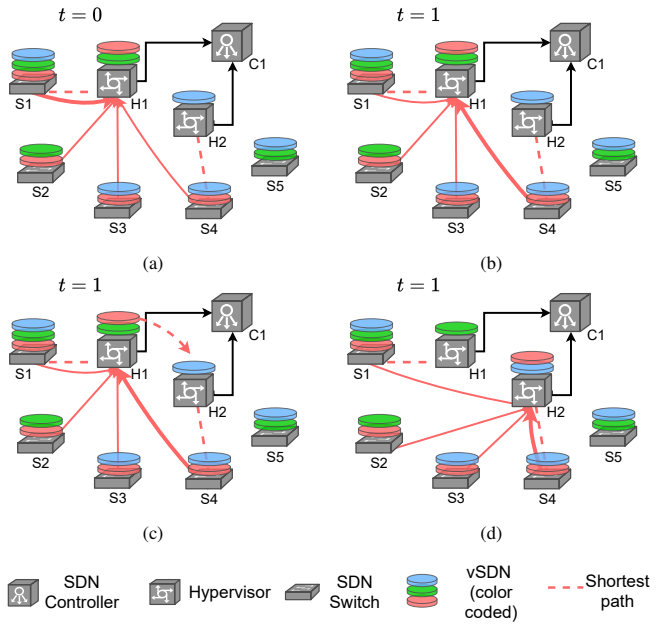


Fig. 1: Migration concept: (a) $t = 0$ initial mapping to hypervisor H1, (b) $t = 1$ variation in flow rate, (c) $t = 1$ find suitable hypervisor, and (d) vSDN migration to H2

of controllers and their locations [5]. Hypervisors in vSDN require optimal placement in the network for efficient virtualization. The hypervisor placement problem (HPP) tackles the question of finding optimal locations for hypervisors. Several placement strategies have been developed for CPP and HPP, to optimize network performance. The majority of the works focus on minimizing the worst-case latency but do not consider the dynamic traffic flows. The control plane traffic distribution or *flows* can be subjected to extreme variations over time. The dynamic nature of traffic leads to prioritize certain flow paths mapping to the least latency path, for the overall performance of the network.

A sample scenario is depicted in Figure 1. The colored discs represent each vSDN and the switches with the same colored disc are part of the same vSDN. The controllers for each vSDN are chosen within the VNs by selecting the node with the least latency over all VNs. For the sake of simplicity, all vSDNs are controlled by the same controller $C1$, with 2 hypervisor instances $H1$ and $H2$. The thickness of the arrow depicts the number of flows from the switch to the hypervisor. For the initial vSDN to hypervisor mapping, the hypervisor closest to the switch carrying a majority of flows for any vSDN is chosen. Let us consider the red vSDN to show our concept. At time $t = 0$, the red vSDN is mapped to $H1$, as switch $S1$ has the majority of flows and is closer to $H1$ (shown in Figure 1(a)). With dynamic traffic, this distribution changes and at $t = 1$, $S4$ has the majority of traffic (Figure 1(b)). Since $S4$ is closer to $H2$, the vSDN migrates from $H1$ to $H2$ (Figure 1(c)), while the controller does not change its location. The increase in latency during this migration phase is added to the overall latency of the network. Since the traffic can suffer from huge variations over time, an optimal method is required to dynamically change the vSDN to hypervisor mapping, while minimizing the overall latency of the system. This is the motivation behind our work.

To improve the network performance, the model considers the effects of load balancing while also minimizing the latency. The optimal dimensioning and placement of hypervisors based on these performance parameters are formulated using novel linear programming models (MILPs). Multiple heuristics are also introduced to reduce the time constraint. The evaluation is based on a quantitative analysis of the optimal solution and comparison with the proposed heuristics. To the best of our knowledge, this is the first work that emphasizes hypervisor processing capacity and changes in control plane traffic. We also consider processing latency into account with propagation latency for latency calculation.

III. Related Work

This section provides an overview of the placement problem origin, existing research in controller and hypervisor placement problem and the hypervisor migration protocol, which forms the basis of our hypervisor placement and reassignment model.

A. Controller Placement Problem

The placement problem is a common linear programming model, derived from the k-center problem and facility location problem (FLP) [6], [7]. CPP was first introduced by Heller et al. [5]. Qin et al. [8] and Xiao et al. [9] consider the controller placement problem for heterogeneous environments. [9] aimed at partitioning the WAN into smaller SDN domains using spectral clustering algorithms.

In [10], the author discusses a Pareto-based optimal placement framework POCO for large SDN networks, which provides users with different placement options based on their performance criteria, such as latency, resilience to failure, and load balancing. [11] provides a load balancing mechanism for already placed controllers using switch groups. Khorramizadeh et al. in [12] developed a heuristic based on obtaining a Pareto set of solutions for controller placement with multiple objective functions. To confront dynamic traffic and node failure conditions, the placement [13] and [14] solves CPP based on controllers' utilization, power consumption, and cost of installation dynamically based on its traffic. Bari et al. [15] provides a detailed description of SDN deployment in a WAN with OpenFlow switches with switch reassignment and dynamic switching of controllers, using a dynamic matching algorithm.

B. Virtualized SDN: Hypervisor Placement Problem

The first work on the hypervisor placement problem is given in [2]. It provides an extensive study of the cost of virtualization in terms of various latency metrics: worst-case latency, average latency, average maximum latency, and maximum average latency. It also considers the diversity of SDN networks with multi-controller SDN switch deployment. It compares a centralized hypervisor architecture with a distributed one, with

Symbol	Description
$G(V, E)$	Physical network substrate
V	Set of physical nodes
E	Set of physical links
M	Set of vSDN requests
C_j	Processing capacity of hypervisor at location j
L_j^t	Load of hypervisor at location j at time t
$vSDN_m(m, R_m^t)$	vSDN request m and message rate R_m^t at time t
$R_{v,m}^t$	Flow rate of VN v for vSDN m at time t
V_m	Set of virtual nodes of $vSDN_m$
q	Number of hypervisors in network
Φ	Set of potential hypervisor locations
$\xi(m, v)$	VNE mapping of VN v of vSDN m
$d(src, dst)$	shortest path between src and dst
c_m	Controller node for $vSDN_m$, $c_m \in V_m$
T	set of timeslots, $t \in T$

TABLE I: Notations definition.

a varying number of hypervisor placements. Killi et al. [1] introduced a joint hypervisor and controller placement strategy. They first propose a controller placement strategy when the hypervisor locations are fixed, which provides the basis of a joint hypervisor controller placement model. The model is designed to minimize the worst-case latency for all vSDN requests and generalizes the model for average latency. In [16], authors considered a dynamic hypervisor placement scenario and formulated an ILP to minimize control plane propagation latency. In [17], the authors proposed a MILP based Joint HPP-CPP for different network latency metrics and give the best geographical arrangements based on user load and latency thresholds. Further, authors in [4] introduced a hypervisor migration protocol and virtualization layer architecture that migrates hypervisor instances without notifying the controller. Existing hypervisor implementations, such as FlowVisor [18], HyperFlex [19], OpenVirteX [20] support both centralized and distributed architecture, but provide static configurations. Deric et al. [21] [22] modeled the hypervisor CPU utilization and processing delays, which have been used in our work.

IV. Problem Formulation

A. System Model

The physical network is modeled as a graph $G(V, E)$ where V represents the nodes and E represents the edges of the network. We consider a fully connected distributed hypervisor architecture. All physical nodes are considered as potential hypervisor locations ($\Phi = V$), and the processing capacity of the hypervisor nodes is given by C_j, where j is the node ID. The processing capacity is defined by the maximum flow rate each hypervisor can handle. A finite time horizon T is defined with $|T|$ timeslots, with $t = \{0, 1, 2...|T|\}$. L_j^t represents the sum of flow rates of the vSDNs assigned to the hypervisor at j. The set of vSDN requests are represented by M, where mth vSDN request is defined by its index m and assigned flow rate R_m^t. V_m represents the set of VNs of vSDN m, where controller node c_m is selected within V_m.

The function $\xi(m, v)$ represents the VNE mapping for each vSDN tenant. For simplifying the model, the VNE mapping is chosen to be random, i.e., each vSDN request's nodes V_m are mapped randomly to the physical substrate nodes V. The model is divided into two sub-models: placement and reassignment, to make the initial placement decision realistic for a given set of static demands. The placement model focuses on optimizing the initial locations of hypervisors with the given flow rates and vSDN requests at time $t = 0$. The reassignment model focuses on vSDN reassignments to existing hypervisor instances. We track variations in processing latency and imbalances in hypervisor load to find reassignments. Table I summarizes all the notations used in this model.

Variable	Description
x_j	= 1, if hypervisor installed at location $j \in \Phi$, 0 otherwise
$\beta_{j,m}^t$	= 1, if vSDN $m \in M$ is assigned to hypervisor at j at time $t \in T$, 0 otherwise
$\beta map_{v,j,m}^t$	= 1, if virtual node $v \in V_m$ is assigned to hypervisor at j at time $t \in T$, 0 otherwise
$\Delta_{\mu,j}^t$	= 1, if physical node $\mu \in V$ is controlled by hypervisor at j at time $t \in T$, 0 otherwise
$\beta mig_{j,j_n,m}^t$	= 1, if vSDN m is reassigned from hypervisor at j to hypervisor at j_n at time $t \in T$, 0 otherwise

TABLE II: Decision variables.

B. Optimization Model Formulation

The model is formulated as a multi-objective MILP model, to minimize end-to-end latency and balance the load on the assigned hypervisors. The variations in vSDN flow rate $R_{v,m}^t$ leads to varying load on each hypervisor L_j^t over each timeslot. The model minimizes the overall latency of the network and triggers reassignments depending on the overhead in latency incurred. The model also tracks the processing load on each hypervisor and prevents hypervisor overload, determined by C_j and L_j^t. Table II lists the decision variables for the MILP. Variable x_j is the only static decision variable that indicates the placement of the hypervisor at location j. All other decision variables of Table II vary over given timeslots t.

Before presenting the MILP model constraints, let us define the three costs considered in the problem: worst-case end-to-end latency, migration cost, and load balancing cost which constitutes the objective functions.

1) The *worst-case end-to-end latency* $Cost_{e2e}^{m,t}$ is calculated as the maximum sum of propagation and processing latency within its VNs.

$$Cost_{e2e}^{m,t} = \max_{v \in V_m} \left\{ (Cost_{prop}^{v,j,m,t} + Cost_{proc}^{v,m,t}) \beta map_{v,j,m}^t \right\} \quad (1)$$

The *propagation latency* $Cost_{prop}^{v,j,m,t}$ for a virtual node v of vSDN m to its controller c_m is calculated as the sum of propagation latency from v to its assigned hypervisor j and the propagation latency from the hypervisor to its controller node. The propagation latency between two nodes s and d, $d(src, dst)$, is calculated from the latency incurred by the shortest path between src and dst. The propagation latency of VN v, from vSDN m, with controller at VN c_m via hypervisor

j is given by

$$Cost_{prop}^{v,j,m,t} = d\Big(\xi(m,v), j\Big) + d\Big(j, \xi(m,c_m)\Big) \tag{2}$$

The *processing latency* $Cost_{prop}^{v,j,m,t}$ is defined as a function of the flow rate of a given VN. The parametric function $f(rate)$ is used to calculate latency from the rate, and is given by the values referenced from [22]. The processing latency for a given timeslot is given by

$$Cost_{proc}^{v,m,t} = f(R_{v,m}^t) \tag{3}$$

2) The *Migration cost* $MCost_{j,j_n,m}^t$ is defined as the maximum latency of the affected packets within all VNs V_m for reassigning the vSDN m. To maintain homogeneity between end-to-end latency costs and migration costs, we model the migration cost in terms of latency as well. The function $g(R_{v,m}^t)$ calculates the average latency incurred by the hypervisor during the migration phase, based on the rate of the migrating vSDN. It has been derived from the experimental values in [4]. The propagation latency between the initial and target hypervisor instances has not been included in the migration costs. The cost for reassigning vSDN m from hypervisor j to j_n at time t, is given by

$$MCost_{j,j_n,m}^t = \max_{v \in V_m} g(R_{v,m}^t) \beta mig_{j,j_n,m}^t \tag{4}$$

3) The *load balancing cost* is defined as the linear imbalance between the maximum and minimum loaded hypervisor instances at a given timeslot. The rates of each vSDN assigned to a hypervisor add to the hypervisor load. The load balancing cost or linear load imbalance is defined as

$$Cost_{lb}^t = \max_{j \in \Phi} L_j^t - \min_{j \in \Phi} L_j^t \beta_{j,m}^t \tag{5}$$

The load L_j^t on any hypervisor j at given time is defined as

$$L_j^t = \sum_{m \in M} R_m^t \beta_{j,m}^t, \ \forall \ j \in \Phi, t \in T \tag{6}$$

Let us now introduce the problem formulation. A hierarchical model for multi-objective optimization is used, which prioritizes the end-to-end latency as the primary objective function, and the load balancing cost as the secondary objective.

$$obj1 : \min \left\{ \frac{1}{|T|} \sum_{t \in T} \left[\frac{1}{|M|} \sum_{m \in M} (Cost_{e2e}^{m,t}) \right] + MCost_{j,j_n,m}^t \right\} \quad \forall m \in M, j, j_n \in \Phi, t \in T \tag{7}$$

$$obj2 : \min \left\{ \frac{1}{|T|} Cost_{lb}^t \right\} \forall t \in T \tag{8}$$

subjected to following constraints:

$$\sum_{j \in \Phi} x_j = q, \ \forall \ t \in T \tag{9}$$

$$\sum_{j \in \Phi} \beta_{j,m}^t = 1, \ \forall \ m \in M, t \in T \tag{10}$$

$$\beta map_{v,j,m}^t \leq \beta_{j,m}^t, \ \forall \ v \in V_m, m \in M, j \in \Phi, t \in T \tag{11}$$

$$x_j \leq \sum_{m \in M} \beta_{j,m}^t, \ \forall \ v \in V_m, m \in M, j \in \Phi, t \in T \tag{12}$$

$$\sum_{v \in V_m} \beta map_{v,j,m}^t \leq |V_m| x_j, \ \forall \ v \in V_m, m \in M, j \in \Phi, t \in T \tag{13}$$

$$\Delta_{\mu,j}^t \leq \sum_{m \in M} \sum_{\substack{v \in V_m \\ \xi(m,v) = \mu}} \beta_{j,m}^t, \forall \ v \in V_m, m \in M, j \in \Phi, t \in T \tag{14}$$

$$\sum_{j \in \Phi} \Delta_{\mu,j}^t \leq 1, \ \forall \ \mu \in V, t \in T \tag{15}$$

$$L_j^t \leq C_j, \ t \in T \tag{16}$$

$$\beta_{j_n,m}^t = \sum_{j \in \Phi} \beta mig_{j_n,j,m}^t, \ \forall \ m \in M, j_n \in \Phi, t \in T \tag{17}$$

$$\beta_{j_n,m}^t = \sum_{j \in \Phi} \beta mig_{j,j_n,m}^{t+1}, \ \forall \ m \in M, j_n \in \Phi, t \in T \tag{18}$$

Constraint (9) fixes the number of installed hypervisors in the network to q. (10) ensures each vSDN is mapped to only one hypervisor for a given time instance. Constraint (11) forces all VNs of every vSDN to be connected to the same hypervisor at each timeslot. (12) forces the hypervisor installation variable x_j to be zero if no vSDN is mapped to it. All virtual nodes of vSDNs assigned to hypervisor j need to be mapped only if the hypervisor instance exists at location j. This is controlled by (13). Constraints (14) and (15) ensure that a hypervisor instance is assigned to control a physical node μ if at least one VN of any vSDN is assigned to the given hypervisor, and the VN is mapped onto this node μ. Hypervisor overload is prevented by constraint (16). The reassignment constraints (17) and (18) use law of conservation technique to ensure mapping and reassignment of each vSDN for all timeslots.

Some of the above constraints overlap for placement and reassignment submodels of the MILP. The proposed MILP is a modified version of the facility location problem, which is NP-hard. Thus, the dynamic hypervisor placement and reassignment model is also an NP-hard problem [6].

The placement model can be defined as:

$$obj1, obj2$$

for time $t = 0$, subjected to constraints (9) to (16), with decision variables referred from Table II. $\beta mig_{j,j_n}^{m,t}$ is not used as no migrations occur at $t = 0$. The following decision variables are used:

$x_j \in \{0,1\} \ \forall \ j \in \Phi$
$\beta_{j,m}^0 \in \{0,1\} \ \forall \ j \in \Phi, m \in M$
$\beta map_{v,j,m}^0 \in \{0,1\} \ \forall \ v \in V_m, j \in \Phi, m \in M$
$\Delta_{\mu,j}^0 \in \{0,1\} \ \forall \ \mu \in V, j \in \Phi$

The optimal solution for placement model is inputted into the reassignment model. The decision constants for $t = 0$ include all optimal values of the decision variables. The reassignment model can be formulated as:

$$obj1, obj2$$

for time $t > 0$, subjected to constraints (10) to (18). The reassignment model optimizes only for $t > 0$. Since x_j is fixed as input for the placement model, it is not included in

Algorithm 1 Initial Greedy Mapping

Input: $G(V, E)$, $m \in M$, q, hypervisor locations hyp_loc
Output: vSDN-hypervisor mapping $mapped[m]\ \forall\ m \in M$

1: **for all** m in M **do**
2: **for** j in hyp_loc **do**
3: find j : $min\{max_{v \in V_m}(d(\xi(m, v), j) + d(j, c_m) + f(R^0_{v,m}))\}$
4: **if** $L^0_j + R^0_m < C_j$ **then**
5: $mapped[m] = j$
6: $L^0_j += R^0_m$
7: **else**
8: goto step 2 with $hyp\_loc - j$

the reassignment model. The decision variables (see Table II) for $t = 0$ are excluded, and are stated below.

$\beta^t_{j,m} \in \{0, 1\}\ \forall\ j \in \Phi, m \in M, t \in T - \{0\}$
$\beta map^t_{v,j,m} \in \{0, 1\}\ \forall\ v \in V_m, j \in \Phi, m \in M, t \in T - \{0\}$
$\Delta^t_{\mu,j} \in \{0, 1\}\ \forall\ \mu \in V, j \in \Phi, t \in T - \{0\}$
$\beta mig^t_{j,j_n,m} \in \{0, 1\}\ \forall\ j \in \Phi, j_n \in \Phi, m \in M, t \in T - \{0\}$

V. Heuristic Algorithms

In this section, we introduce our proposed algorithms for the dynamic HPP and reassignment model. We also present rudimentary heuristics which are used for evaluation. In the proposed MILP model, q hypervisors are placed among V nodes, creating $\binom{V}{q}$ possible placement solutions. For each hypervisor instance, $|M|$ vSDNs are mapped, the solution space increases by $|M|^q$. The reassignments can occur at any time instance, and all vSDNs can migrate to any of $q - 1$ hypervisor instances (or q for simplicity). Thus, over T timeslots, the model creates a $\binom{V}{q} * |M|^{2qT}$ solution space. This solution space is large, even for small networks like Abilene. The heuristic approach aids in reducing this complexity and aims at finding a solution closer to the optimal solution, while relaxing the time constraint. Similar to the MILP, the proposed heuristics are also divided into placement and reassignment heuristics.

A. Placement Heuristics

The placement heuristics are divided into two phases: location selection and vSDN mapping. We present two hypervisor location selection heuristics: **Random** and **Centrality based**. **Random** selects random hypervisor locations based on a number of hypervisors q. In **Centrality based** selection, network nodes are sorted with the highest betweenness centrality, and the q highest nodes are selected as hypervisors locations. The vSDN to hypervisor mapping heuristic is a simple greedy algorithm, which maps each vSDN to the hypervisor with minimum worst-case end-to-end latency, while also considers hypervisor overload.

Algorithm 1 shows the pseudo-code for initial mapping of vSDNs. hyp_loc represents the set of hypervisor locations selected using the random or centrality-based heuristics. The

Algorithm 2 GGLM

Input: Initial random population P_0, max iterations $Itermax$

1: **while** $i < Itermax$ **do**
2: **for** Set S in P_i **do**
3: Apply algo 1
4: Calculate fitness score for S
5: Select parents from P_i, add to P_{i+1}
6: Crossover and create offsprings for P_{i+1}
7: Mutate if enabled
Output: Converged Solution

vSDN requests are sorted with decreasing flow rates, giving the higher priority to map the vSDN with larger rates. The algorithm looks for the hypervisor which provides minimum worst-case end-to-end latency cost $Cost^{m,t}_{e2e}$ for given vSDN m. The algorithm selects the hypervisor j with minimum end-to-end latency for the vSDN m, from the set hyp_loc. If the selected hypervisor j overloads when vSDN m is mapped, j is removed from potential hypervisor locations and the algorithm runs recursively until a suitable mapping is found. The initial rate of the mapped vSDN is added to the load of selected hypervisor. The traffic model (refer to Section VI) ensures no vSDN remains unmapped.

We also introduce an evolutionary algorithm for combined node selection and vSDN mapping, called **Genetic Algorithm with Greedy load mapping (GGLM)**. The proposed combined node selection and placement heuristic are presented in Algorithm 2. The genetic algorithm (GA) is part of a class of stochastic evolutionary heuristic algorithm, based on a natural selection and mutation mechanism. We discuss each component of the GA separately below:

- **Initial Population and Representation:** The population is created as a binary string of size equal to the number of nodes in the network, and '1' represents the node selected as the hypervisor. The initial population consists of 4 chromosomes, where each chromosome is created randomly, and the number of '1's is fixed to q.
- **Fitness Function:** A weighted sum of end-to-end latency cost and load balancing cost is formulated as the fitness function, and is given by

$$FitnessFunction : \alpha \sum_{m \in M} \frac{Cost^{m,0}_{e2e}}{|M|} + \gamma Cost^0_{lb} \tag{19}$$

The weights α and γ are set to normalize each cost term and provide equal weightage to both objectives. The fitness score is calculated for each chromosome, and two chromosomes with the least fitness score are selected as parents for the next iteration, as the objective is for minimization.
- **Crossover:** The selected parents are crossed over for two new offsprings, which form the population for the next iteration. A partial crossover technique is used to ensure each offspring also conforms to the number of hypervisors in every iteration. Figure 2 illustrates the partial crossover technique. The parent chromosomes have $\{3, 5, 6, 10, 11\}$ and

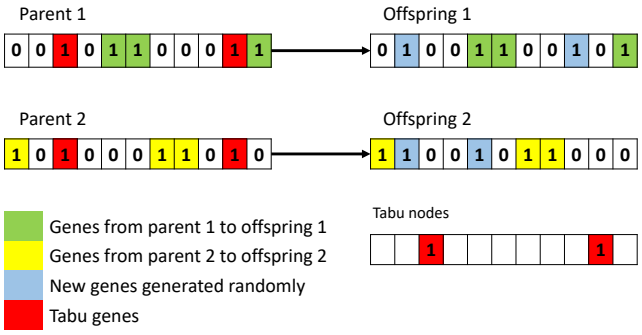


Fig. 2: Partial Matching Crossover

$\{1, 3, 7, 8, 10\}$ the bits turned on, representing hypervisor locations at respective indices of the network graph. The exchange arrays for each parent are $\{5, 6, 11\}$ and $\{1, 7, 8\}$ respectively. Consequently, the exchange arrays give the locations for offspring 1 and offspring 2, respectively. Since both parents have $\{3, 10\}$ locations in common, using them will lead to repetition in population, as the heuristic will get stuck to a local maxima/minima, instead of reaching the optimal location matrix. The common locations are considered as *tabu* for both offsprings, and the remaining locations are decided randomly, ensuring that offspring 1 does not have any location already installed, or any location from the tabu list.

- **Mutation:** A mutation is used to diversify the population by altering one more gene in the offsprings. The mutation probability is set to 0.05. After each mutation, a random index is selected for a reverse mutation, for keeping the number of hypervisor installations constant for the next iteration.

B. Reassignment Heuristics

The reassignment needs to occur at runtime, implying the applied heuristics should be simple. To solve the dynamic changes in control plane traffic, i.e., flow rate, a simple heuristic can provide solutions with minimum information and the least number of information processing steps. The reassignment model tracks variation in rates and calculates $Cost^{m,t}_{e2e}$ for every timeslot. A reassignment occurs whenever an increase in latency is detected due to flow rate distribution, or hypervisor overload. Algorithm 3 shows the pseudo-code for reassignment heuristic when only considering flow rate variations.

The reassignment heuristic in the case of hypervisor overload is shown in Algorithm 4. In some instances, reassigning a single vSDN does not affect hypervisor overload. The algorithm lists all vSDNs mapped to the overloaded hypervisor and recursively reassigns vSDNs to the hypervisor with the lowest load until its load conforms to hypervisor capacity. The target hypervisor does not suffer overload during this reassignment phase, as all vSDNs with lower rates are reassigned, ensuring the target hypervisor is not overloaded. Due to sorting of virtual networks connected by a particular hypervisor, and remapping vSDNs to the least loaded hypervisors, a balanced load is achieved over time.

Algorithm 3 Reassignment Model

Input: vSDN to hypervisor mapping $mapped[m]$ at time t, hypervisor locations hyp_loc, hypervisor loads L^t_j, worst case e2e latency $Cost^{m,t}_{e2e}$
Output: new $mapped[m]$
for m in M where $mapped[m] = h$ **do**
  if $Cost^{m,t}_{e2e} > Cost^{m,t-1}_{e2e}$ **then**
    Find $j_n = h$: $min(Cost^{m,t}_{e2e})$
    if $MCost^t_{j,j_n,m} < Cost^{m,t}_{e2e} - Cost^{m,t-1}_{e2e}$ **then**
      $mapped[m] = j_n$
      Increase overall latency by $MCost^t_{j,h_n,m}$ and update L^t_j and $L^t_{j_n}$

Algorithm 4 Hypervisor Overload

Input: vSDN to hypervisor mapping $mapped[m]$, hypervisor locations hyp_loc, hypervisor loads L^t_j
Output: new vSDN to hypervisor mapping $mapped[m]$
1: **for all** j in hyp_loc **do**
2: $j_n = h : \min(L^t_h)$
3: **for** sorted $m \in M$ and $mapped[m] = j$ by R^t_m **do**
4: **if** $L^t_j > C_j$ **then**
5: $mapped[m] = j_n$
6: Increase overall latency by $MCost^t_{j,h_n,m}$ and update L^t_j and $L^t_{j_n}$

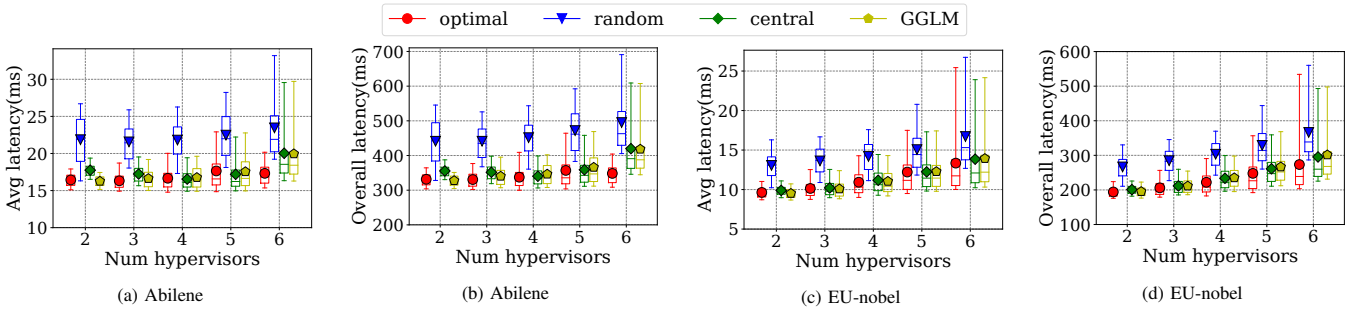
The algorithm 3 is referred as *Balanced Migration* (BM) in performance evaluation. This heuristic is compared against a greedy heuristic, *Greedy migration* (GM). GM reassigns vSDN whenever an increase in $Cost^{m,t}_{e2e}$ is observed, irrespective of the migration cost $MCost^t_{j,j_n,m}$. Algorithm 4 is triggered in both migration heuristics, as hypervisor overload can lead to loss of data in the network.

VI. Performance Evaluation

The proposed MILP and heuristics are evaluated with input ranges for each system variable, illustrated in Table III. Two topologies are used: Abilene topology consists of 12 nodes and 15 bidirectional edges and EU-Nobel topology comprises 28 nodes and 41 bidirectional edges. The hypervisor processing capacity (C_j) is limited to 1000 $msgs/s$ and the rates of each vSDN are defined by the number of vSDNs $(|M|)$, their maximum rate $(max(rate_m))$ and the number of hypervisors installed (q). The maximum assigned rate of each vSDN is defined as

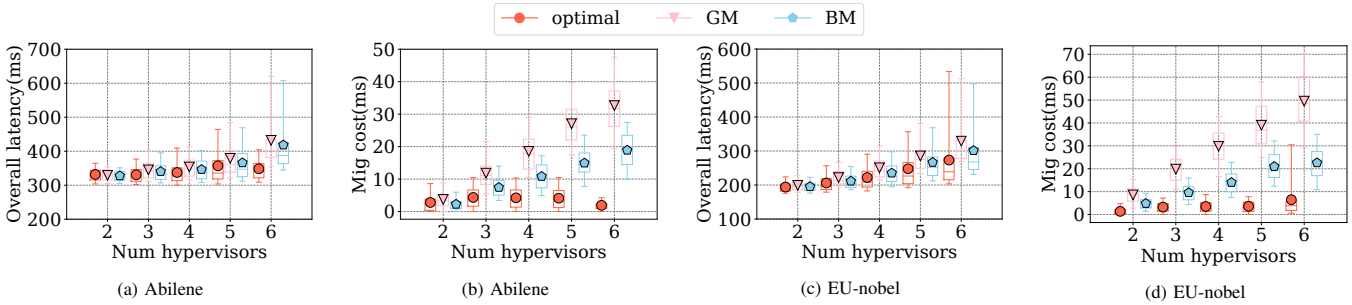
$$max(rate_m) = \frac{C_j * q}{|M|} \tag{20}$$

The maximum vSDN rate ensures each hypervisor has a proportionate load, irrespective of the number of hypervisors in the network. This also triggers hypervisor overload in the network and triggers vSDN reassignments. All vSDNs are initialized with a random rate, ranging between 0.3 to 0.5 times the maximum allowed rate. The number of virtual nodes of every vSDN request is uniformly generated between 3 and 6. The controller for each vSDN is selected with the minimum average controller-to-VN propagation latency. The process-



(a) Abilene (b) Abilene (c) EU-nobel (d) EU-nobel

Fig. 3: Comparison of performance of optimal solution vs placement heuristics in terms of average and overall latencies on Abilene and EU-nobel topologies.



(a) Abilene (b) Abilene (c) EU-nobel (d) EU-nobel

Fig. 4: Comparison of performance of optimal solution vs migration heuristics in terms of overall latency and migration cost on Abilene and EU-Nobel topologies.

ing latency function is a uniform linear distribution, with a maximum of 500 ms latency for a flow rate of 600 $msgs/s$. A Poisson arrival process is applied independently for each vSDN, with interarrival times for each process defined by the parameter λ. For $|M| = 10$, half of the vSDNs are randomly selected for a more aggressive increment pattern, indicating demanding networks. For a higher number of vSDNs, 20% are selected as the more demanding networks, with $\lambda = 0.6$ for less demanding networks, and 0.8 for demanding networks. These values were selected to find the optimal arrival rates to achieve statistically evident results. The flow rate profile adheres to the assigned vSDN rate: the sum of flows from all VNs of any vSDN cannot be more than the assigned rate of the vSDN at any given timeslot. The evaluations are performed on a server with an 8 core GenuineIntel Common KVM processor running at 2.5 GHz with a memory of 90 GB. The MILP and heuristics are implemented in Python and optimized using Gurobi Optimizer. Every setup is run for 30 iterations and 95% confidence intervals are used to get statistically reliable results. In this paper, we present three parameters for evaluation: i) *Average latency* is the average of worst-case end-to-end latency for all vSDNs over all timeslots; (ii) *Migration cost* is the sum of the migration costs over all timeslots to get the latencies incurred due to migration in the given time horizon; and (iii) *Overall latency* is the sum of routing and migration latencies for all timeslots. To reduce the number of dimensions involved, results with a varying number of hypervisors are shown.

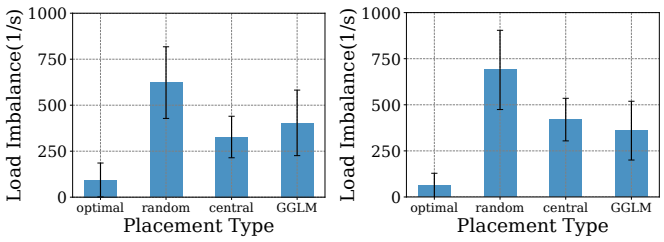
The increase in average and overall latency with more hypervisors is attributed to the traffic model, as the maximum vSDN rate is proportional to the number of hypervisors installed in the network. This leads to higher processing latency, even when the propagation latency decreases with an increase in hypervisors.

Parameter	Values
Network Topology	Abilene, EU-Nobel
No. of hypervisors	2, 3, 4, 5, 6
No. of vSDNs	20, 30, 40, 50, 60
No. of virtual nodes per vSDN	U(3,6)
Initial vSDN rate	0.3 to 0.5 of $max(rate_m)$
Rate increase λ_{inc}	0.6 - 0.8
Processing latency(VN size 3 to 6)	U(200,600) ms
No. of iterations	30

TABLE III: Distribution of parameters for the evaluation

A. *Impact of Placement Strategies*

From the plots in Figure 3, the random placement strategy incurs the highest average and overall latencies for both topologies. The average latency increases with the number of hypervisors with a similar slope as the optimal solution. In the case of a varying number of vSDNs, higher latencies are observed with this placement strategy. The difference in latency parameters is higher for Abilene than EU-Nobel. The latencies shown with the central nodes placement strategy follow closely with our proposed algorithm GGLM for Abilene. In the case of EU-Nobel, it can be observed that GGLM performs better, and follows the optimal solution closely. It can be asserted that the proposed algorithm shows better performance for larger networks. The placement strategies have little to no effect on



(a) (b)

Fig. 5: Performance of placement strategies for load balancing at time $t = 0$ for (a) Abilene and (b) EU-Nobel topologies.

the migrations in the network. The various migration strategies are discussed separately.

B. Impact of Migration Strategies

As shown in Figure 4, the migration cost increases with an increase in the number of hypervisors. Due to the presence of more hypervisors, more suitable hypervisor instances are available to each vSDN when an increase in the end-to-end latency is observed. This leads to an increase in the number of migrations with the least contribution to minimizing the average latency. A decrease in migration cost is observed with an increase in the number of vSDNs. Due to the design parameters of vSDN flow rates, the higher number of vSDNs have lower rates. This causes the migration cost for any vSDN to be lower, triggering more migrations, with a decrease in migration cost.

The heuristic provides higher migration costs than the optimal solution, resulting in a higher overall latency than the optimal solution. It is observed that the number of migrations is higher for the optimal solution, while the cost of migration is lower. The MILP considers the vSDN for the lowest flow rate for migration, while the heuristic considers two criteria for vSDN migration: hypervisor overload and increase in average latency per timeslot per vSDN. In the case of hypervisor overload, the heuristic migrates the vSDN with the lowest rates, assigned to the overloaded hypervisor, to the least loaded hypervisor until the hypervisor load is below its capacity. In the case of latency-based migration, the vSDNs with higher rates may also migrate, contributing to higher migration costs. For the Abilene topology, the GM heuristic shows a slightly higher number of migrations than the BM heuristic but suffers from higher overall latency and migration cost. In the case of EU-Nobel, the gap between the two heuristics is increased, concluding better performance of BM for larger networks.

C. Analysis of Heuristics- Impact of Load Balancing

The load imbalance measure for heuristics is calculated as the average of the difference of loads between the maximum loaded and minimum loaded hypervisor instances. It does not depict the variations over time. The MILP tries to minimize the load imbalance for all timeslots. The proposed heuristics do not perform dynamic load balancing per timeslot. The main focus is to prevent hypervisor overload, and trigger migrations whenever a hypervisor instance is overloaded. Furthermore, all migrations due to an increase in latency consider hypervisor overload instead of load balancing. This results in a higher load imbalance overall timeslots for each placement and migration strategy. Both migration strategies perform similarly for different parameters, with the BM strategy showing lower load imbalance for EU-Nobel topology.

To perform a fair comparison of the heuristics and optimal solution in terms of load balancing, we consider only the initial load mapping. Figure 5 shows the load imbalance for the optimal solution with different placement strategies. The random placement displays the highest load imbalance. The placement based on the centrality and our proposed algorithm, GGLM, show similar performance in load balancing. The GGLM shows better performance for EU-Nobel topology. In GGLM, the latency and load balancing objectives are combined in a normalized weighted sum for multi-objective optimization. For the same weights for both objectives, a high load imbalance was observed. To improve the performance of the second objective, the weight of the load balancing objective is increased to 0.65 and latency weight to 0.35. These values do not include the normalization factors of the cost terms in Equation (19). The altered weights showed an improvement in load balancing with no degradation in the latency objective. All results shown for the GGLM heuristic use the same weights.

Number of decision variables	$\sum_{m \in M} \lvert V \rvert * \lvert T \rvert + \sum_{m \in M} \lvert V \rvert * \lvert V_m \rvert * \lvert T \rvert + \lvert V \rvert + 2 * \lvert V \rvert^2 * \lvert T \rvert + \lvert V \rvert^2 * \lvert M \rvert * \lvert T \rvert$
Number of equality constraints	$\lvert V \rvert + \lvert M \rvert * \lvert T \rvert + \lvert V \rvert * \lvert M \rvert * \lvert T \rvert$
Number of inequality constraints	$\sum_{m \in M} \lvert V \rvert * \lvert V_m \rvert * \lvert T \rvert + \lvert V \rvert + \lvert V \rvert * \lvert M \rvert * \lvert T \rvert + \lvert V \rvert^2 * \lvert T \rvert + 2 * \lvert V \rvert * \lvert T \rvert$
Number of general constraints	$2 * \lvert V \rvert * \lvert T \rvert$

TABLE IV: Complexity Analysis of MILP

D. Complexity Analysis

The MILP model for combined placement and reassignment is based on capacitated facility location problem (CFLP), which is an NP-hard problem. The Gurobi optimizer uses a dual simplex method, which can traverse in the worst case, all 2^n nodes in the search space for an optimal solution. This makes the order of complexity of finding the optimal solution to be $O(2^n)$ where n is the number of solution points in the search space. The heuristic approaches reduce the complexity of the problem significantly, by providing sub-optimal but acceptable solutions. The complexity of the proposed MILP is presented in Table IV in terms of the number of decision variables and different constraints used.

E. Runtime Analysis

The average completion times of the models for the heuristics and the optimal solution is given in Table V. The heuristics perform almost 100x faster than the MILP solved with 5% gap. The proposed algorithm includes genetic algorithm as the placement strategy, which contributes the maximum in

Topology	Solution type	Average runtime(s)
Abilene	MILP	248.20
	Random greedy	1.32
	Central greedy	1.22
	GGLM	16.9
EU-Nobel	MILP	645.45
	Random greedy	1.49
	Central greedy	1.33
	GGLM	19.82

TABLE V: Comparison of runtime of MILP and heuristics.

runtime. The genetic algorithm, due to its stochastic nature, takes the longest to reach convergence. The runtime of GGLM can be reduced by using a subset of locations for hypervisor placement or reducing the number of iteration to reach convergence. It was observed that selecting the nodes with the highest centrality is the best measure for small networks, while the placement using GGLM shows better performance for both objective functions. The average runtimes for Abilene and EU-Nobel topologies for the GGLM heuristic are $16.9s$ and $19.82s$, respectively. This implies the heuristic offers scalability for large physical networks and a higher number of virtual networks.

VII. Conclusion

This research aims to design a hypervisor placement scheme in a dynamic virtualized SDN environment to minimize the end-to-end control plane latency. The model shows that vSDNs can be dynamically assigned in a distributed hypervisor architecture, taking migration costs into account. The presented model considers migration cost in terms of latency to depict the network overhead incurred by each virtual network. The placement scheme shows an increase in latency with an increase in the number of hypervisors. Furthermore, the evaluation shows that the proposed heuristics provide latency measures close to the optimal solution. The heuristic provides higher migration costs than the optimal solution. In conclusion, the presented model opens a new facet of dynamic placement and virtual network reassignment in virtualized networks.

As part of future work, heterogeneity can be introduced, with hypervisors of different capacities, different sizes of virtual networks, etc. Moreover, the dynamic hypervisor placement model can be extended to a joint controller and hypervisor placement, with increased dynamicity by enabling hypervisor and controller migrations.

Acknowledgment

This work has been funded by the Germany Federal Ministry of Education and Research under project AI-NET ANTILLAS (project ID #16KIS1318).

References

[1] B. P. R. Killi and S. V. Rao. On placement of hypervisors and controllers in virtualized software defined network. *IEEE Transactions on Network and Service Management*, 15(2):840–853, 2018.

[2] A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer. Control Plane Latency With SDN Network Hypervisors: The Cost of Virtualization. *IEEE Commun. Surveys Tuts.*, 18(1):665–685, 2016.

[3] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer. Survey on network virtualization hypervisors for software defined networking. *IEEE Commun. Surveys Tuts.*, 18(1):665–685, 2016.

[4] A. Basta, A. Blenk, H. Belhaj Hassine, and W. Kellerer. Towards a dynamic sdn virtualization layer: Control path migration protocol. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 354–359, 2015.

[5] B. Heller, R. Sherwood, and N. McKeown. The controller placement problem. *ACM SIGCOMM Comput. Commun. Rev.*, 42(4):473–478, 2012.

[6] Ling-Yun Wu, Xiang-Sun Zhang, and Ju-Liang Zhang. Capacitated facility location problem with general setup cost. *Computers and Operations Research*, 33(5):1226–1241, 2006.

[7] A. M. C. Hörhammer. Dynamic hub location problems with single allocation and multiple capacity levels. In *2014 47th Hawaii International Conference on System Sciences*, pages 994–1003, 2014.

[8] Q. Qin, K. Poularakis, G. Iosifidis, and L. Tassiulas. SDN controller placement at the edge: Optimizing delay and overheads. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 684–692, 2018.

[9] P. Xiao, W. Qu, H. Qi, Z. Li, and Y. Xu. The SDN controller placement problem for WAN. In *2014 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 220–224, 2014.

[10] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann. Heuristic approaches to the controller placement problem in large scale sdn networks. *IEEE Transactions on Network and Service Management*, 12(1):4–17, 2015.

[11] Y. Zhou, Y. Wang, J. Yu, J. Ba, and S. Zhang. Load balancing for multiple controllers in sdn based on switches group. In *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 227–230, 2017.

[12] Mostafa Khorramizadeh and Vahid Ahmadi. Capacity and load-aware software-defined network controller placement in heterogeneous environments. *Computer Communications*, 129:226 – 247, 2018.

[13] M. T. I. ul Huque, W. Si, G. Jourjon, and V. Gramoli. Large-scale dynamic controller placement. *IEEE Transactions on Network and Service Management*, 14(1):63–76, 2017.

[14] Y. Liu, H. Gu, X. Yu, and J. Zhou. Dynamic SDN controller placement in elastic optical datacenter networks. In *2018 Asia Communications and Photonics Conference (ACP)*, pages 1–3, 2018.

[15] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba. Dynamic controller provisioning in software defined networks. In *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, pages 18–25, 2013.

[16] Sen Chen, Weiqiang Sun, and Weisheng Hu. On dynamic hypervisor placement in virtualized software defined networks (vsdns). In *2020 22nd International Conference on Transparent Optical Networks (ICTON)*, pages 1–5, 2020.

[17] Deborsi Basu, Abhishek Jain, Uttam Ghosh, and Raja Datta. Flexarch: Flexible controller placement architecture for hypervisor assisted vsdn-enabled 5g networks. In *2020 IEEE Globecom Workshops (GC Wkshps*, pages 1–6, 2020.

[18] R. Sherwood et al. Flowvisor: A network virtualization layer. In *OpenFlow Consortium, Palo Alto, CA, USA,*, volume OPENFLOW-TR-2009-1, pages 222–227, 2009.

[19] A. Blenk, A. Basta, and W. Kellerer. Hyperflex: An sdn virtualization architecture with flexible hypervisor function allocation. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 397–405, 2015.

[20] Ali Al-Shabibi, Marc De Leenheer, Matteo Gerola, Ayaka Koshibe, William Snow, and Guru Parulkar. Openvirtex: A network hypervisor. In *Open Networking Summit 2014 (ONS 2014)*, Santa Clara, CA, March 2014. USENIX Association.

[21] Nemanja Derić, Amir Varasteh, Arsany Basta, Andreas Blenk, and Wolfgang Kellerer. Sdn hypervisors: How much does topology abstraction matter? In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 328–332, 2018.

[22] N. Derić, A. Varasteh, A. Van Bemten, A. Blenk, and W. Kellerer. Enabling SDN hypervisor provisioning through accurate CPU utilization prediction. *IEEE Transactions on Network and Service Management*, pages 1–1, 2021.