Resilient Control Plane Design for Virtualized 6G Core Networks

Ferenc Mogyorósi, Péter Babarczi, *Senior Member, IEEE*, Johannes Zerwas, *Graduate Student Member, IEEE*, Andreas Blenk, and Alija Pašić

***Abstract*—With the advent of 6G and its mission-critical and tactile Internet applications running in a virtualized environment on the same physical infrastructure, even the shortest service disruptions have severe consequences for thousands of users. Therefore, the network hypervisors, which enable such virtualization, should tolerate failures or be able to adapt to sudden traffic fluctuations instantaneously, i.e., should be well-prepared for such unpredictable environmental changes. In this paper, we propose a latency-aware dual hypervisor placement and control path design method, which protects against single-link and hypervisor failures and is ready for unknown future changes. We prove that finding the minimum number of hypervisors is not only NP-hard, but also hard to approximate. We propose optimal and heuristic algorithms to solve the problem. We conduct thorough simulations to demonstrate the efficiency of our method on real-world optical topologies, and show that with an appropriately selected representative set of possible future requests, we are not only able to approach the maximum possible acceptance ratio but also able to mitigate the need of frequent hypervisor migrations for most realistic latency constraints.**

***Index Terms*—Software defined networks, virtual networks, resilient hypervisor placement, intelligent algorithms.**

Manuscript received 19 November 2021; revised 29 April 2022; accepted 8 July 2022. Date of publication 22 July 2022; date of current version 12 October 2022. The work of F. Mogyorósi and A. Pašić was supported by the ÚNKP-21-3 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund, and by the KDP-2021 Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund. This work was supported by Projects no. 134604, no. 137698 and no. 128062 that have been implemented with the support provided by the National Research, Development and Innovation Fund of Hungary, financed under the FK_20, PD_21, and K_18 funding schemes, respectively. The work of A. Pašić was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences. This work received funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 438892507. The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany (BMBF) in the programme of "Souverän. Digital. Vernetzt." joint project 6G-life, project identification number 16KISK002. The associate editor coordinating the review of this article and approving it for publication was M. Tornatore. *(Corresponding author: Ferenc Mogyorósi.)*

Ferenc Mogyorósi, Péter Babarczi, and Alija Pašić are with the Department of Telecommunications and Media Informatics, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, 1111 Budapest, Hungary, and also with the ELKH-BME Information Systems Research Group, 1117 Budapest, Hungary (e-mail: mogyorosi@tmit.bme.hu; babarczi@tmit.bme.hu; pasic@tmit.bme.hu).

Johannes Zerwas is with the Chair of Communication Networks, Technical University of Munich, 80290 Munich, Germany (e-mail: johannes.zerwas@tum.de)

Andreas Blenk is with Siemens AG, 81739 Munich, Germany (e-mail: andreas.blenk@siemens.com).

Digital Object Identifier 10.1109/TNSM.2022.3193241

I. Introduction

FUTURE 6G applications demand a flexible and adaptable control plane design providing low-latency [1]–[3]. Such applications range from electronic health, tele-presence and holographic communications, industrial automation towards smart environments with augmented or virtual reality. In such cyber-physical applications, 6G networks are expected to be integrated as a mission-critical component. This requires 6G networks to go even beyond the reliability and resilience as provided by 5G ultra-reliable low-latency communications [4]–[6]. Additionally, it is envisioned that applications can run isolated on one physical infrastructure using the concept of end-to-end network slicing. Hence, slicing the 6G core network is an integral part of future networks [7]. In order to realize such slicing and virtual control, a virtualization layer is needed. This virtualization layer is realized by so-called network hypervisors [8].

In a nutshell, the network hypervisors sit between the Software-Defined Network (SDN) switch and the tenant (application) controllers. They intercept and modify control plane messages in order to abstract the physical topology and isolate controllers of different tenants. The tenant controllers can only operate on their respective slice – virtual SDN (vSDN) – of the physical network. As a result, the SDN control path extends and spans over the application controller, the virtualization layer, and the assigned slice of the network.

In order to meet the stringent demands of the applications, the virtualization layer itself needs to be reliable and also to be able to adapt to sudden (load) changes in the environment [9]. Distributed control and virtualization layers have become the default approach. However, distributed designs introduce the question of how many hypervisor instances are needed and where to place them. Existing solutions to design and dimension such layers cover a wide range of optimizations w.r.t. to latency, e.g., minimizing latency [10], providing Quality of Service (QoS) guarantees [11], balancing the load in the virtualization layer [12] or optimizing for resilience [13].

Dynamic load scenarios are addressed by migrating hypervisor instances to other locations or assigning switches to other hypervisor instances [12], [14]. This re-active approach is no longer suited given the anticipated tight delay constraints of future 6G applications. Also optimization of hypervisor placements with respect to resilience currently only covers single link failures but leaves out failures of hypervisor instances [13].

In this paper we present a dual hypervisor placement and a switch-to-hypervisor assignment method. The protection-based approach creates two control paths via different hypervisor instances for each slice to control the vSDNs in the network. The algorithm leverages knowledge from past requests – where available – and optimizes against a representative set of future demands. Thereby, it generates a prepared placement that alleviates the need for reconfigurations, such as migrations of hypervisor instances, in case of sudden demand changes or failures. Our particular contributions are as follows:

- We introduce a dual hypervisor placement problem with edge-disjoint control paths to protect against single link and hypervisor failures.
- In our latency-aware design both paths meet the latency requirement of the vSDN request, thus, enable hypervisor migration without any QoS degradation.
- We minimize the number of hypervisors [13] while maximizing preparedness, i.e., leaving enough reaction possibilities open to unknown failures and vSDN requests with the application of representative request sets.
- We prove that minimizing the number of hypervisors is NP-hard and hard to approximate, and propose a heuristic and an Integer Linear Program (ILP) to solve it.

The rest of the paper is organized as follows. Section II summarizes the related work on resilient control plane design. We formulate our latency-aware resilient hypervisor placement problem in Section III. We prove that approximating the optimal number of hypervisors is as hard as set cover in Section IV, and propose a data structure which is leveraged in our greedy heuristic and ILP formulation. Given the above hypervisor number, Section V introduces an optimal method for maximizing acceptance ratio of a given request set, and discusses the generation of representative sets to prepare the initial hypervisor placement for future changes. Finally, we present the experimental results in Section VI and conclude the paper in Section VII.

II. Related Work

A. Self-Stabilizing Distributed In-Band Control Plane

In [18], the authors argue that in order to provide high availability for connections in SDN, the *control plane must be distributed*. Furthermore, despite data and control packets arrive at the same port, *in-band control* is desired in these networks to leverage the network's high path diversity instead of operating and maintaining a dedicated expensive (and sparse) control network. In the introduced self-stabilizing control plane, automatic topology discovery and management of controllers is possible in a distributed manner. Furthermore, it provides a switch-to-controller assignment and quickly establishes in-band control paths between the control- and data-plane and between the distributed controller instances, even after adding or removing controllers or after edge failures. Renaissance [19] deals with the design of a reliable in-band and distributed control plane which tolerates concurrent controller, link and communication failures. The proposed algorithms ensure that every switch is managed by an operational controller all the time, moreover, the switch can be reached within a bounded communication delay after a bounded number of failures.

In vSDNs, the control paths from each switch to the virtual controller must traverse a network hypervisor instance, which besides abstraction of physical network resources provides the isolation of both data- and control plane traffic of different tenants. In [8], different software- and hardware-based hypervisor implementation were enumerated, and presented a high variety of possible distributed vSDN control plane architectures, e.g., flexible control plane virtualization techniques [20] or using multi-controller switches [15]. In [13], resilient (edge-disjoint) in-band control paths were designed between the virtual switches and the virtual controller, traversing the single hypervisor instance responsible for the given switch. In our control plane operation, we build on the above concepts, which enable a distributed resilient in-band control plane.

B. Network Preparedness: Maximizing Future Options

Preparedness for unseen challenges in the future is a desired property in communication networks and has been thoroughly investigated in the literature [21]–[25]. Preparedness could range from being resilient against edge- or node-failures (i.e., being fault-tolerant [13], [18], [19]), through timely response to disaster alerts [14] to adaptation to changing traffic patterns [23], [26]. Although the proposed solutions to these problems significantly differ, they share the same design principle: 1) prepare a computationally tractable model of future challenges, and 2) propose a metric which can measure how well-prepared the network is against them.

Traditional optimization methods rely on finely-tuned objective functions which are carefully tailored to the actual parameters of the problem, and owing to the huge number of options cannot incorporate all future possibilities in a tractable manner. Robust and stochastic optimization [21] make a step towards resolving this issue, and already can handle some sort of uncertainty in the input parameters or provide a solution without exact knowledge of the (future) inputs. Similarly to optimization, machine learning, e.g., reinforcement learning algorithms still require an external reward system and often a specific training set on which the algorithm can rely on [27].

In [25], an information-theoretic tool-set, called *empowerment* was proposed, where no external reward system is provided for the agent – relies only on its own observations –, which distinguishes it from machine learning and traditional optimization problems. In a discrete setting, the metric returns the number of different states (its logarithm to be precise) the network can adapt to in response to challenges, which was later applied to reconfigurable networks [28]. Future freedom of action can be modeled as a physical force as well which tries to maximize the entropy of the system [22], and was applied to several use cases.

In networking, minimum interference routing shows a huge resemblance with this principle, as through appropriately selected link weights it keeps bottleneck resources open as long as possible [23], i.e., it maximizes future acceptable requests without explicitly being told to do so. A less powerful version of this approach was applied to resilient hypervisor

TABLE I
COMPARISON OF HYPERVISOR PLACEMENT METHODS

vSDN requests	Placement method	Optimized locations	Min. objective	Constraint	Preparedness
All known	k-HPP [15]	controller and hypervisor	max/avg latency	# hypervisor	-
	Joint Placement (JHCP) [10]	joint controller/hypervisor	max latency	# hypervisor	-
Deployed set	Dynamic HPP (ILP) [16]	hypervisor	max/avg latency	# hypervisor	hypervisor migration (traffic load)
	Dynamic HPP (Genetic) [12]	hypervisor	max latency & load	# hypervisor	hypervisor migration (traffic load)
	Dynamic HPP (MILP) [14]	hypervisor	max latency	migration time	hypervisor migration (disaster alerts)
Unspecified	Resilient HPP [13]	joint controller/hypervisor	# hypervisor	max latency	single-link failure + migration
	Betweenness-based HPP [17]	controller and hypervisor	betweenness centrality	-	-
	Latency-Aware Resilient HPP	joint controller/hypervisor	# hypervisor	max latency	single-link/node failure + migration

placement in vSDNs [13]. Without the knowledge of such intrinsic motivation which drives the network towards maximum future options, the proposed solution leaves multiple placements available from which the network was able to select in a self-driving manner. In this paper, we significantly improve this model with the application of representative request sets.

C. Hypervisor Placement Problem

Similarly to other placement problems (e.g., SDN controller placement [29]), finding appropriate hypervisor locations boils down to the mathematical task of facility location [30]. However, in the *Hypervisor Placement Problem (HPP)*, additional requirements and constraints must be considered. We summarize the main directions in Table I.

Several works [10], [15], [31] investigate the static version of the HPP. Here, all virtual networks – set of virtual switches and their controllers – are given as input to the placement algorithm. The task is to find a hypervisor placement which minimizes the maximum or average latency both for all control paths and per individual virtual network. Note that, these formulations are not appropriate for dynamically arriving (future, thus unknown) vSDN requests, nor can they handle failures in the network. Furthermore, owing to the fact that all requests are considered in their ILP at the same time, it has an excessive running time, thus, it is not applicable for frequently changing and/or thousands of vSDN requests [13]. In [11], the latency requirement of 5G and future 6G network were considered in the controller and hypervisor placement problem, and the proposed multi-objective optimization algorithm was able to minimize the propagation latency of network function demands and improve QoS. Dobrijevic *et al.* [17] provide an availability analysis of the static HPP under node and link failures. They consider placements without protection and three protection-based versions. Their placement, however, is solely based on the betweenness centrality of the possible locations and does not provide any guarantees regarding latency or load.

In contrast, [12], [16], [27] introduce a reactive approach called *Dynamic Hypervisor Placement Problem* (DHPP): the hypervisor placement as well as the switch to hypervisor assignments are redesigned in order to balance load and react to fluctuating demands. However, adaptation is only considered in reactive manner. Chen *et al.* [16] propose an ILP to minimize control path latency and take the migration overhead of hypervisors into account. Their approach lowers the migration cost. Amjad *et al.* [12] specifically focus on load balancing within a distributed network virtualization layer. The DHPP is also considered in the context of resilience: the embedding of virtual networks and control planes might be redesigned in reaction to disaster alerts [14]. The authors propose two models in [14], where 1) the hypervisor-to-switch assignment is fixed but the hypervisor instance can be migrated to a safe location, and 2) the hypervisor locations are fixed but the switches can be reassigned to different hypervisor instances. However, the temporal disruption of the control path and the transient behavior of the network during migration of the (single) hypervisor instance responsible for a given switch are not considered in these works [14], [27].

None of the aforementioned works maximizes the reaction possibilities of the network pro-actively through an intelligent algorithm design or adds backup hypervisors to lower the severity of disruptions due to migrations. Such an approach will be discussed in this paper. A first step was made towards a well-prepared control plane in [13] where the authors investigated the *Resilient Hypervisor Placement Problem* providing pre-allocated link-disjoint switch-to-hypervisor and hypervisor-to-controller control paths between the physical switch, virtual controller and the corresponding hypervisor instance(s), respectively. Although there are multiple hypervisor locations with a running hypervisor instance [13], the proposed architecture does not consider any backup hypervisors and each switch is controlled by a single hypervisor instance only.

III. PROBLEM FORMULATION

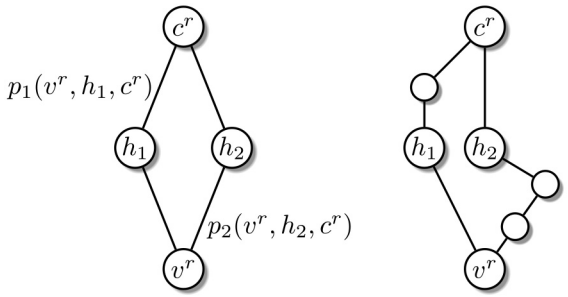
We argue that a pro-active design of backup SDN control paths and hypervisors is necessary if we desire to meet the strict recovery time requirements declared in the Service Level Agreement (SLA) of the vSDN requests [13], [32]–[36]. Similarly to previous approaches on control plane design we do *not consider edge capacities* in our formulations [15], [29]–[31], [37], i.e., control paths of different switches can be calculated as independent sub-problems. Therefore, in our intelligent problem formulation, we focus on the number and location of hypervisor instances shared by multiple switches and vSDNs.

TABLE II
NOTATION FOR THE PHYSICAL SDN NETWORK $\mathcal{G}$

Notation	Description
$\mathcal{G}(\mathcal{V},\mathcal{E})$	Physical SDN network graph
$\mathcal{V}$	Set of physical locations (network nodes) $v \in \mathcal{V}$
$\mathcal{E}$	Set of physical network edges $e \in \mathcal{E}$
$\mathcal{S}$	Set of physical swtich locations with $\mathcal{S} \subseteq \mathcal{V}$
$\mathcal{H}$	Set of potential hypervisor locations with $\mathcal{H} \subseteq \mathcal{V}$
$\mathcal{C}$	Set of potential controller locations with $\mathcal{C} \subseteq \mathcal{V}$
$l(e)$	Latency of edge e, with $l(e) \in \mathbb{R}^+$
$\mathcal{P}(s,t)$	Set of simple paths between locations s and t
$d(p_i)$	Latency of path $p_i \in \mathcal{P}(s,t)$: $d(p_i) = \sum_{e \in p_i} l(e)$
L	Maximum global control path latency constraint
$\mathcal{P}(s,h,c)$	Set of paths between nodes s and c passing through node h with $\forall p_i \in \mathcal{P}(s,h,c) : d(p_i) \leq L$

TABLE III
NOTATION FOR THE VIRTUAL SDN (VSDN) REQUESTS $\mathcal{R}$

Notation	Description
$\mathcal{R}$	Set of vSDN requests $r \in \mathcal{R}$
$\mathcal{V}^r$	Set of virtual network nodes of vSDN request r, $r \in \mathcal{R}$
v^r	Virtual network node $v^r \in \mathcal{V}^r$
L^r	Maximum control latency for vSDN request r, $r \in \mathcal{R}$
c^r	Virtual controller node of vSDN request r, $r \in \mathcal{R}$
$\mathcal{H}_{v^r}$	Hypervisor nodes $\{h_1, h_2\}$ of virtual network node v^r



(a) Logical view (b) Physical view

Fig. 1. Edge-disjoint control paths $p_1(v^r, h_1, c^r)$ and $p_2(v^r, h_2, c^r)$ between the switch virtual controller traversing the corresponding hypervisors.

A. Physical SDN Network Model

The network topology is modeled as a graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ with nodes (locations) $v \in \mathcal{V}$ connected by undirected edges $e \in \mathcal{E}$. Each location might host a physical SDN switch $\mathcal{S} \subseteq \mathcal{V}$ (usually $\mathcal{S} = \mathcal{V}$), while potential hypervisor and controller locations are given in the sets $\mathcal{H} \subseteq \mathcal{V}$ and $\mathcal{C} \subseteq \mathcal{V}$, respectively. The latency $l(e)$ of an edge e is computed from the geographical distance between the two network nodes that are connected via edge e (i.e., the capacity of the edge is not considered in our control plane design), which is used for evaluating the latency of end-to-end network paths. The set $\mathcal{P}(s,t)$ contains simple paths between network nodes s and t. The i-th simple path in $\mathcal{P}(s,t)$ is denoted as $p_i(s,t)$. When the end nodes of the path are irrelevant or clear from the context, we will simply use p_i. The total latency of path $p_i(s,t)$ is $d(p_i(s,t)) = \sum_{e \in p_i(s,t)} l(e)$. Table II summarizes the notations used for the physical SDN network.

B. Virtual SDN Requests

Our focus in this paper is resilient control plane design. Therefore, we assume that the data plane (virtual SDN nodes and virtual links) is already given and embedded to a set of physical SDN nodes by an arbitrary embedding algorithm. Accordingly, a vSDN request $r \in \mathcal{R}$ is defined by the set of SDN nodes $\mathcal{V}^r \subseteq \mathcal{V}$. In contrast to [10], [15], [31] where the tenant's requested topology is considered as part of the input of the joint control- and data-plane design, this set contains the virtual switches of the tenant's request, as well as possibly some additional (intermediate) physical nodes as the results of virtual (data-plane) edge embeddings spanning multiple physical hops. Hence, without loss of generality, we assume that $\mathcal{V}^r$ is a connected subgraph of the network. Furthermore, each request has a given latency constraint L^r declared in the SLA which all control paths have to satisfy. Table III summarizes the notation for the vSDN requests $\mathcal{R}$.

C. Resilient Virtual Control Plane

In Figure 1, we present the considered control plane architecture in this paper. In order to satisfy the SLA, all nodes in $v^r \in \mathcal{V}^r$ have to be controlled by the request's virtual controller[1] c^r within the specified latency, i.e., $\forall v^r \in \mathcal{V}^r : d(p(v^r, c^r)) \leq L^r$. Furthermore, in a control plane resilient against single edge failures and prepared for hypervisor migration, all vSDN nodes $v^r \in \mathcal{V}^r$ of a request r have to be connected to their corresponding controller location c^r through edge-disjoint paths (both paths satisfying the latency constraint) traversing hypervisor instances $\mathcal{H}_{v^r} = \{h_1, h_2\}$, where h_1 and h_2 denotes the two (primary and backup) hypervisor assigned to v^r, respectively. Note that, a backup hypervisor for a given switch can be a primary for another one and vice versa, which significantly lowers the overhead of our solution.

Opposed to [15], [31], the controller location c^r is not specified in advance, i.e., not part of the input vSDN request r. It can be chosen from the set $\mathcal{C}$ to satisfy the latency requirement of the vSDN request[2] depending on the switch to hypervisor assignment, which improves latency [10] and enables self-driving operation [13]. For a fair comparison of different design methods, we will assume in this paper that $\forall r \in \mathcal{R} : L^r = L$, where L is the *maximum global control path latency constraint* for each request.

D. Latency-Aware Resilient Hypervisor Placement Problem

Without controllers (without vSDN requests) the resilient hypervisor placement problem is essentially a modified facility location problem on the physical network topology where each customer must be served by two facilities that reach them

[1] In this paper we assume that an embedded vSDN request is operated by a single virtual controller. However, our model can be easily extended to multiple controllers per request in the future.

[2] Without loss of generality, we assume that $\mathcal{C}$ is the same set of controller locations for all vSDN requests.

on disjoint paths. In our case, the hypervisors can be viewed as facilities and the physical switches as customers. In [30], the authors considered two special cases of the "cover-by-pairs" optimization problem that arise when facilities need to be placed so that each customer is served by two facilities with reaching them on two disjoint paths. We will shortly present the definitions of the pathwise-disjoint problem [30] and modify them to our resilient hypervisor placement problem by extending them with latency constraints and possible controller locations as follows.

In [30], a strongly connected graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ is given, together with sets $\mathcal{S} \subseteq \mathcal{H} \subseteq \mathcal{V}$, where $\mathcal{S}$ is the set of customer (switch) locations and $\mathcal{H}$ is the set of potential facility (hypervisor) locations. For each pair (s,h), $s \in \mathcal{S}$ and $h \in \mathcal{H}$ set $\mathcal{P}(s,h)$ contains simple (and also shortest in [30]) paths from s to h in $\mathcal{G}$, which is nonempty by the strong connectivity assumption.

Definition 1: Suppose $s \in \mathcal{S}$ is a customer location and $\{h_1,h_2\}$ is a pair of potential facility locations in $\mathcal{H}\backslash\{s\}$. We say that **pair $\{h_1,h_2\}$ covers s in a pathwise-disjoint fashion** if there exist paths $p_1 \in \mathcal{P}(s,h_1)$ and $p_2 \in \mathcal{P}(s,h_2)$ that are edge-disjoint.

Next, we can generalize this definition from a single customer to all $s \in \mathcal{S}$ in network $\mathcal{G}$.

Definition 2: A subset $\mathcal{H}' \subseteq \mathcal{H}$ is called a **pathwise-disjoint cover for** $\mathcal{S}$ if (1) $\forall s \in \mathcal{S}\backslash\mathcal{H}'$ there is a pair $\{h_1,h_2\} \subseteq \mathcal{H}'\backslash\{s\}$ such that $\{h_1,h_2\}$ cover s in a pathwise-disjoint fashion, or (2) $\forall s \in \mathcal{H}'$ the facility at s covers itself.

In order to extend the above facility location definitions of [30] and apply them to our hypervisor placement problem, we drop the $\mathcal{S} \subseteq \mathcal{H}$ requirement, i.e., $\mathcal{S}$ and $\mathcal{H}$ can be arbitrary subsets of $\mathcal{V}$. Moreover, we assume that edge lengths $l(e)$ on $\mathcal{G}(\mathcal{V},\mathcal{E})$, possible controller locations $c \in \mathcal{C}$ and latency requirement L are given. Let $\mathcal{P}(s,h,c)$ denote the set of (not necessary simple) paths from s to c traversing node h *satisfiing the latency requirement* L, i.e., $\forall p \in \mathcal{P}(s,h,c) : d(p) \leq L$.

Definition 3: Suppose $s \in \mathcal{S}$ is a switch location and $\{h_1,h_2\}$ is a pair of potential hypervisor locations in $\mathcal{H}\backslash\{s\}$. We say that **pair $\{h_1,h_2\}$ can control s in a latency-aware pathwise-disjoint fashion** if there exists $c \in \mathcal{C}$ for which paths $p_1 \in \mathcal{P}(s,h_1,c)$ and $p_2 \in \mathcal{P}(s,h_2,c)$ are edge-disjoint.

Next, we can generalize this definition from a single switch to all $s \in \mathcal{S}$ in network $\mathcal{G}$.

Definition 4: A subset $\mathcal{H}' \subseteq \mathcal{H}$ is called a **latency-aware pathwise-disjoint control cover for** $\mathcal{S}$ if (1) $\forall s \in \mathcal{S}\backslash\mathcal{H}'$ there is a pair $\{h_1,h_2\} \subseteq \mathcal{H}'\backslash\{s\}$ such that $\{h_1,h_2\}$ can control s in a latency-aware pathwise-disjoint fashion, or (2) $\forall s \in \mathcal{H}'$ the hypervisor at s can control that switch.

Note that, the latter condition does not harm the resilience of our approach even if s is assigned only to a single hypervisor, as there is no physical edge between the two which can fail. Moreover, the unavailability of that location disrupts both the switch and the hypervisor. However, we need to handle the lack of backup hypervisor for these switches when the hypervisor should be migrated to a new location.

Finally, we define the main problem investigated in this paper. Our goal is to provide the required level of resilience at minimal operational cost, which is ultimately the goal of the provider. Fewer hypervisors are cheaper to deploy and operate, and that is the main goal in addition to meeting the SLA requirements.

Problem 1 [Latency-Aware Resilient Hypervisor Placement Problem (LHPP)]: Given $\mathcal{G}(\mathcal{V},\mathcal{E})$, sets $\mathcal{S} \subseteq \mathcal{V}, \mathcal{H} \subseteq \mathcal{V}, \mathcal{C} \subseteq \mathcal{V}$ and maximum global control path latency L, find a latency-aware pathwise-disjoint control cover $\mathcal{H}' \subseteq \mathcal{H}$ for $\mathcal{S}$ where the number of hypervisors $|\mathcal{H}'|$ is minimal.

The result of an (optimization) algorithm for Problem 1 is the set of selected hypervisor locations $\mathcal{H}' \subseteq \mathcal{H}$ and the primary and backup switch-to-hypervisor assignment of the switches $\forall s \in \mathcal{S}\backslash\mathcal{H}' : \mathcal{H}_s = \{h_1,h_2\} \subseteq \mathcal{H}'$.

E. Preparedness of Different Hypervisor Placements

Our goal in this paper is to find a hypervisor placement and switch-to-hypervisor assignment which is well-prepared for future vSDN requests. By "preparedness" we mean that our initial placement is capable of serving a variety of requests, but is also prepared to being migrated in response to serve dynamically arriving vSDN requests. The required control-plane reconfiguration can be initiated periodically in a self-driving manner by the network to adapt to day-night or hourly load shifts. Note that in these control-plane reconfigurations, only the hypervisors are relocated, while the controllers and vSDNs remain in the same place.

In addition, the network can continuously monitor its own performance by calculating the gap between the current and optimal hypervisor locations calculated for the current vSDN requests. If the gap between the current and the optimal solution is larger than a certain threshold (e.g., 5%), the network should initiate reconfiguration and hypervisor migration immediately, or at the end of a certain period if slotted operation is assumed. In the latter case, the time slots and thus the frequency of hypervisor migrations can be adapted to the dynamics of the network either by the network operator or by the network itself in a self-driving manner.

Another important aspect of dynamic network management is the hypervisor migration process itself, which is crucial for providing a certain QoS. Our proposed architecture in Figure 1 with two redundant hypervisor instances provides great flexibility in the reconfiguration process compared to approaches using a single hypervisor per switch [12], [14], [16], [27]: during migration one of the instances can be fixed while the second instance is moved (transferring the primary role between the two instances if necessary). After state synchronization with the new instance it can be promoted as primary hypervisor and the old instance can be migrated if necessary. However, a thorough analysis of different strategies is out of the scope of the paper.

IV. Latency-Aware Resilient Hypervisor Placement Algorithms

In this section, we investigate the computational complexity and propose different solution possibilities for our resilient control plane design problem. In Section IV-A, we prove that the LHPP formulated in Problem 1 is not only NP-hard but

also as hard to approximate as set cover. We introduce an efficient representation of hypervisor locations which can provide latency-aware pathwise-disjoint control covers and discuss the computational cost of path and set pre-computations in Section IV-B. Section IV-C contains our algorithmic solution for the problem using greedy set cover. Finally, Section IV-D provides our ILP formulation for LHPP to minimize the number of hypervisors in the latency-aware pathwise-disjoint control cover for dynamically arriving vSDN requests.

A. Computational Complextiy and Approximability of LHPP

It was shown that pathwise-disjoint facility location [30] and set cover by pairs [38] are not only NP-hard but also are at least as hard to approximate as Set Cover (SC). Therefore, they cannot be approximated within $(1-\epsilon)\ln n$ for any $\epsilon > 0$ unless $P = NP$ [39]. We will prove the same inapproximability result for LHPP as well.

Theorem 1: LHPP is NP-hard and at least as hard to approximate as SC.

Proof: In a nutshell, we prove the theorem by converting an arbitrary instance of SC into an equivalent instance of LHPP in polynomial-time whose optimal solution differs from the optimal solution of the SC instance by 1. Thus, any approximation factor for our problem would imply a same factor for SC as well.

Given an arbitrary instance of SC which consists of a ground set $\mathcal{S} = \{s_1, \ldots, s_n\}$ and a collection $\mathcal{U} = \{U_1, \ldots, U_m\}$ of subsets of S, for which without loss of generality $\mathcal{S} = \cup_{U \in \mathcal{U}} U$. In the transformed instance of LHPP, the graph $\mathcal{G}'(\mathcal{V}', \mathcal{E}')$ has the following nodes in $\mathcal{V}'$:

1) switch nodes $s_i \in \mathcal{S}$ for $i = 1, \ldots, n$,
2) auxiliary nodes x_0 and x_i for $i = 1, \ldots, n$,
3) a single controller location $\mathcal{C} = \{c_0\}$,
4) and potential hypervisor locations h_j for each $U_j \in \mathcal{U}$ and h_0 which doesn't correspond to any member of $\mathcal{U}$, resulting in $\mathcal{H} = \cup_j \{h_j\}$, $j = 0, \ldots, m$.

We add the following undirected edges to $\mathcal{E}'$ in the transformed graph, with latency $\forall e \in \mathcal{E}' : l(e) = 1$:

1) edges connecting switches and auxiliary nodes: $\{s_i, x_0\}$ and $\{s_i, x_i\}$, for $i = 1, \ldots, n$,
2) edges connecting auxiliary nodes and hypervisor locations: $\{x_0, h_0\}$ and $\{x_i, h_j\}$ if $s_i \in U_j$, for $i = 1, \ldots, n$, $j = 0, \ldots, m$,
3) edges connecting the controller location with hypervisor locations: $\{c_0, h_j\}$, for $j = 0, \ldots, m$.

We set the global latency constraint $L = 4$. The constructed graph[3] is given in Figure 2. For all switch s_i there is a path $p_1(s_i, x_0, h_0, c_0)$ with $d(p_1) = 3$ to c_0 through x_0 and h_0. Furthermore, $\forall i, j : s_i \in U_j$ there are further paths $p_2(s_i, x_i, h_j, c_0)$ through x_i with $d(p_2) = 3$. All the other paths from s_i to c_0 through an arbitrary h node have a latency $d(p) \geq 5$. Consequently, setting the latency constraint of the LHPP to $L = 4$ will eliminate those longer paths, leaving p_1 and one of the p_2 paths as the only edge-disjoint path-pair

[3]Note that, a similar transformation was given for the pathwise facility location problem in [30]. However, it is not directly applicable for LHPP.

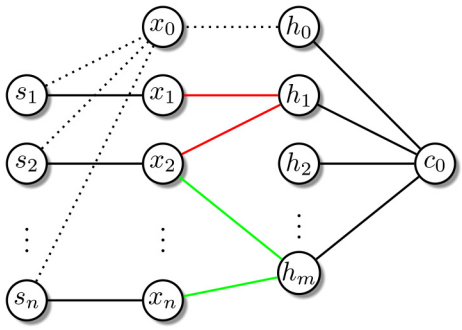


Fig. 2. Polynomial-time transformation of an arbitrary set cover instance into a corresponding instance of LHPP. In the example red and green edges represent that $s_1, s_2 \in U_1$ and $s_2, s_n \in U_m$, respectively.

which can provide a latency-aware pathwise-disjoint control cover for any $s_i \in \mathcal{S}$ in the above LHPP instance.

In order to conclude our proof, we have to demonstrate that the optimal solution of the above LHPP instance is exactly one more than the optimal solution for the SC instance.

(SC $\Rightarrow$ LHPP): Assume that $\mathcal{U}^*$ is a feasible solution to SC. In this case, one can see that $\{h_0\} \cup \{h_j : U_j \in \mathcal{U}^*\}$ is a feasible solution for LHPP, as each $s_i \in \mathcal{S}$ is connected to c_0 through h_0 and h_j such that $s_i \in U_j$. Note that, such h_j exists because $\mathcal{U}^*$ is a cover for $\mathcal{S}$. Owing to the polynomial-time construction $\mathcal{H}_{s_i} = \{h_0, h_j\}$ can control s_i in a latency-aware pathwise-disjoint fashion. Hence, the LHPP solution has one more hypervisors (i.e., h_0) than the number of sets in the SC instance.

(SC $\Leftarrow$ LHPP): Assume that $\mathcal{H}' \subseteq \mathcal{H}$ is a feasible solution for our constructed LHPP instance. For each switch node $s_i \in \mathcal{S}$, there is at least one hypervisor in $\mathcal{H}'$ that does not equal to h_0, thus it can reach c_0 through $h_j \in \mathcal{H}' - \{h_0\}$. Hence, the correspdonding U_j sets for $h_j \in \mathcal{H}' - \{h_0\}$ provides a set cover for $\mathcal{S}$. Again, the LHPP solution has one more hypervisor than the number of sets in the SC instance.

Therefore, LHPP and SC have solution at the same time which differs by one, which proves the NP-hardness of Problem 1. Furthermore, a polynomial-time algorithm with any approximation guarantee for LHPP would yield one for SC as well, which proves the second part of our claim. ■

B. Representing Latency-Aware Pathwise-Disjoint Covers

We introduce a representation of the control architecture shown in Figure 1 without the a priori knowledge of the vSDN request set. We store the possible locations satisfying Definition 3 from which the feasible hypervisor placements for every physical switch $s \in \mathcal{S}$ in the network can be efficiently queried. Hence, we define quartets (c, h_1, h_2, s), where $c \in \mathcal{C}$, $h_1, h_2 \in \mathcal{H}$, and $s \in \mathcal{S}$, which represent that the maximum global latency constraint L is met by controller location $c \in \mathcal{C}$ with edge-disjoint paths $p_1 \in \mathcal{P}(s, h_1, c)$ and $p_2 \in \mathcal{P}(s, h_2, c)$. We define $\mathcal{Q}$ as the set of all quartets that have a disjoint control path-pair satisfying the latency requirement, formally:

$$\mathcal{Q} = \{(c, h_1, h_2, s) | \exists p_1 \in \mathcal{P}(s, h_1, c), p_2 \in \mathcal{P}(s, h_2, c) : p_1, p_2 \text{ are edge-disjoint, and } d(p_1) \leq L, d(p_2) \leq L\}.$$

One can observe that the quartets in $\mathcal{Q}$ corresponding to s contain all possible controller and hypervisor locations for that node which satisfy the latency constraint L. Note that, $\mathcal{Q}$ contains (s, s, s, s) for each $s \in \mathcal{H}$ and possibly other quartets satisfying Definition 3 for $s \in \mathcal{S}$ where $h_1 \neq h_2$. However, important to note that in practice $\mathcal{P}(s, t)$ contains only a predefined number of P simple paths for each (s, t) pair; thus, the concatenation of path in $\mathcal{P}(s, h)$ and $\mathcal{P}(h, c)$ to $\mathcal{P}(s, h, c)$ will contain a limited number of paths as well. Therefore, if path number P is not carefully selected and not large enough, it is possible that $\mathcal{Q}$ will not contain all feasible quartets. Please refer to Section VI-A for further analysis of this question.

The projections of $\mathcal{Q}$ to different dimensions can be efficiently leveraged in the algorithms proposed for LHPP. For example, if we select s and fix the hypervisor pair $h_1, h_2 \in \mathcal{H}$, then we can obtain from $\mathcal{Q}$ all possible controller locations for s for that primary and backup hypervisor pair. Owing to its importance in our LHPP algorithms, we define set $\mathcal{T}$ from the projection of $\mathcal{Q}$ which represent that $\{h_1, h_2\} \in \mathcal{H}$ is a primary and backup hypervisor pair for $s \in \mathcal{S}$ which possibly can provide latency-aware pathwise-disjoint cover for a (unknown) vSDN request in the future, formally:

$$\mathcal{T} = \{(h_1, h_2, s) | \exists c \in \mathcal{C} : (c, h_1, h_2, s) \in \mathcal{Q}\}.$$

As a result, $\mathcal{T}$ contains all hypervisor pairs which can cover $\forall s \in \mathcal{S}$ in a latency-aware pathwise-disjoint fashion with an appropriately selected controller location, denoted as $\mathcal{T}(s)$ for a specific switch s.

Lemma 1: Pre-calculating sets $\mathcal{T}$ and $\mathcal{Q}$ require $O(|\mathcal{V}|^6 \cdot P^4)$ steps, where $P = \max_{i,j} |\mathcal{P}(v_i, v_j)|$.

Proof: In order to construct sets $\mathcal{P}(s, h_i, c)$, as a first step we find the P-shortest simple paths $\mathcal{P}(v_i, v_j)$ (if exist) between all node-pairs in the network topology in $O(|\mathcal{V}|^3 \cdot P)$ steps [40]. Next, we need to check each path-combination from $p_1 \in \mathcal{P}(s, h_i)$ and $p_2 \in \mathcal{P}(h_i, c)$ whether their concatenation satisfy the global latency requirement L or not, which can be done in one step for each P^2 path-pair since their latency is already given by the path generation algorithm. If $d(p_1) + d(p_2) \leq L$ we add the resulting (possibly non-simple) $p(s, h_i, c)$ path to $\mathcal{P}(s, h_i, c)$. As a result, $|\mathcal{P}(s, h_i, c)| \leq P^2$.

Calculating $\mathcal{Q} = \{(c, h_1, h_2, s)\}$ requires to check at most $|\mathcal{C}| \cdot |\mathcal{H}|^2 \cdot |\mathcal{S}|$ times whether $p_1 \in \mathcal{P}(s, h_1, c)$ and $p_2 \in \mathcal{P}(s, h_2, c)$ is edge-disjoint or not, which can be done in linear-time in the number of edges $|\mathcal{E}| \leq |\mathcal{V}|^2$. As there are at most $P^2 \cdot P^2$ number of p_1, p_2 path-pairs and at most $|\mathcal{V}|^4$ quartets (c, h_1, h_2, s) in $\mathcal{Q}$, the total complexity of the quartet generation is $O(|\mathcal{V}|^6 \cdot P^4)$.

Generating $\mathcal{T}$ and $\mathcal{T}(s)$ from $\mathcal{Q}$ can be done even by a naive implementation by checking all $O(|\mathcal{V}|^4)$ elements of $\mathcal{Q}$. Therefore, the overall complexity of the pre-processing is dominated by the $\mathcal{Q}$ generation, resulting in an overall complexity of $O(|\mathcal{V}|^6 \cdot P^4)$. ∎

Although an expensive process (please, refer to Table VI for concrete values), we need to make the above pre-calculation only once for the physical topology, as owing to the high-priority of control plane traffic edge latency values are not influenced by traffic fluctuations of the data plane. Furthermore, in the simulations in Section VI we set P to a small fix constant (i.e., $P = 16$), therefore the pre-computation time is mainly affected by the number of nodes – $O(|\mathcal{V}|^6)$ – in the network.

Algorithm 1: Greedy Heuristic Algorithm for LHPP

Input:
$\forall s \in \mathcal{S} : \mathcal{T}(s) = \{(h_1, h_2)\}$ - set of feasible hypervisor pairs for s;
Output: $\mathcal{H}'$ - hypervisor locations;
$\forall s \in \mathcal{S} : \mathcal{H}_s = \{h_1, h_2\}$ - switch-to-hypervisor assignment;

1 Initialize $\mathcal{H}' := \emptyset$;
`// Phase 1: Perform greedy set cover`
2 **while** $\exists s \in \mathcal{S}$ *not covered by* $\mathcal{H}'$ **do**
3 Find $h^* \in \mathcal{H} \backslash \mathcal{H}'$ for which $\mathcal{H}' \cup h^*$ covers most uncovered switches according to sets $\forall i : \mathcal{T}(s_i)$;
4 Add h^* to hypervisors $\mathcal{H}' := \mathcal{H}' \cup h^*$;
`// Phase 2: Post-processing`
5 **for** $h \in \mathcal{H}'$ **do**
6 **if** $\mathcal{H}' \backslash \{h\}$ *is a cover for* $\mathcal{S}$ **then**
7 $\mathcal{H}' := \mathcal{H}' \backslash \{h\}$;
`// Phase 3: Switch-to-hypervisor assignment`
8 Select $\mathcal{H}_s = \{h_1, h_2\} \in \mathcal{H}'$ for every switch $s \in \mathcal{S} \backslash \mathcal{H}'$ where $(h_1, h_2) \in \mathcal{T}(s)$;

C. Greedy Heuristic Approach

Owing to the high computational complexity of Problem 1, in this section we propose a greedy heuristic for LHPP, summarized in Algorithm 1. The algorithm takes the pre-calculated $\mathcal{T}(s)$ sets as input, and returns the set of hypervisors $\mathcal{H}'$ and switch-to-hypervisor assignment $\forall s \in \mathcal{S} : \mathcal{H}_s = \{h_1, h_2\}$ that cover $\mathcal{S}$ in a latency-aware pathwise-disjoint fashion.

In Phase 1 initially, the cover $\mathcal{H}'$ is an empty set, and in Step 2, we perform a greedy set cover [41] as long as $\mathcal{H}'$ does not cover all $s \in \mathcal{S}$ (or no solution exists): find $h^* \in \mathcal{H} \backslash \mathcal{H}'$ such that $\mathcal{H}' \cup \{h^*\}$ provides a latency-aware pathwise-disjoint control cover for most $s \in \mathcal{S}$ which did not have one previously, formally $\nexists h_i, h_j \in \mathcal{H}' : (h_i, h_j) \in \mathcal{T}(s)$ but $\exists h_i \in \mathcal{H}' : (h_i, h^*) \in \mathcal{T}(s)$. Hypervisor locations at switches $s \in \mathcal{H} \cap \mathcal{S}$ will cover at least one switch – i.e., $\{(s, s)\} \in \mathcal{T}(s)$ – if selected. If there are multiple h^* locations with the maximum number of newly covered switches, then we select one of them randomly. As we add a new hypervisor in each step, the algorithm will terminate in at most $|\mathcal{H}|$ iterations. In Phase 2 we adopt the post-processing phase of the greedy algorithm from [30] in Step 5 to check whether we can remove some locations from $\mathcal{H}'$ while maintaining the latency-aware pathwise-disjoint control cover for every switch. As proposed in [30], we run Algorithm 1 400 times and then one solution is randomly selected from the ones with minimal $|\mathcal{H}'|$. Finally, in Phase 3 we assign a hypervisor pair for every switch $s \in$

$\mathcal{S}\backslash\mathcal{H}'$, and if multiple options available we choose the one with minimal average switch-to-hypervisor latency.

Lemma 2: The time-complexity of Algorithm 1 is $O(|\mathcal{S}| \cdot |\mathcal{H}|^3)$.

Proof: One can observe that we have at most $|\mathcal{H}|$ iterations in Step 2 of Phase 1 (each element of $\mathcal{H}$ is added to $\mathcal{H}'$). In each iteration we have to calculate the number of newly covered switches by every hypervisor location $h \in \mathcal{H}\backslash\mathcal{H}'$, which requires to check the $\mathcal{T}(s_i)$ sets for each uncovered switch ($\forall s_i \in \mathcal{S} : |\mathcal{T}(s_i)| \leq |\mathcal{H}|^2$), resulting in $|\mathcal{S}| \cdot |\mathcal{H}|^2$ operations per iteration and $O(|\mathcal{S}| \cdot |\mathcal{H}|^3)$ complexity for Phase 1 altogether. Post-processing in Phase 2 requires to check the same sets and properties as in Phase 1; thus, it has the same number of operations in worst case. Finally, selecting hypervisor pairs from $\mathcal{T}(s)$ for every $s \in \mathcal{S}\backslash\mathcal{H}'$ in Phase 3 requires $O(|\mathcal{S}| \cdot |\mathcal{H}|^2)$ time. Therefore, Algorithm 1 runs in polynomial-time $O(|\mathcal{S}| \cdot |\mathcal{H}|^3)$. ∎

Assuming that both $|\mathcal{H}|$ and $|\mathcal{S}|$ is in the order of $|\mathcal{V}|$, Lemma 2 gives $O(|\mathcal{V}|^4)$ complexity. In this setting the greedy heuristic proposed for the pathwise facility location problem [30] requires at least $O(|\mathcal{V}|^3)$ steps (no latency constraint considered), while the hypervisor placement with set cover [13] takes $O(|V|^2(|E| + |V|\log_2|V|))$ steps (only single-link failure resilience ensured), thus, we pay a linear complexity increase for a latency-aware resilient placement in worst case even with the usage of the pre-calculated $\mathcal{T}(s)$ sets. However, in practice the $|\mathcal{T}(s)| \ll |\mathcal{V}|^2$ set sizes are small for reasonable global latency constraints L, resulting in an efficient algorithm. We also note that greedy set cover approximates the optimal number of sets within a factor of $\sum_{i=1}^{|\mathcal{V}|} 1/i \leq \ln|\mathcal{V}| + 1$ for the general unweighted case [41], [42], where $|\mathcal{V}|$ refers to the maximum size of any set. Although this approximation factor was successfully transformed for hypervisor placement with greedy set cover in [13], owing to the non-monotonicity of uncovered sets according to the $\mathcal{T}(s)$ values in Step 2 the factor is not transferable for Algorithm 1.

D. Integer Linear Program for LHPP

In order to find the *minimum number of hypervisors* k in a latency-aware pathwise disjoint cover, in this section, we present a mixed integer linear program for LHPP denoted as ILP_k. The required parameters and binary decision variables are summarized in Table IV.

The variable w_h determines whether a hypervisor is located at the network node $h \in \mathcal{H}$. Note that, after having solved the model, the variables w_h specify the optimal set $\mathcal{H}^*$ of hypervisor nodes, specifically, $\mathcal{H}^* = \{h \in \mathcal{H} : w_h = 1\}$. The variable $x_{h,s}$ is set to one if switch $s \in \mathcal{S}$ is controlled by the hypervisor instance placed at node $h \in \mathcal{H}$. Similarly, the variable $y_{h_1,h_2,s}$ is set to one if switch $s \in \mathcal{S}$ is controlled by the hypervisor instances placed at nodes $h_1, h_2 \in \mathcal{H}$ (including variables $y_{s,s,s}$). Our objective is to minimize the number of hypervisors:

$$\min \sum_{h\in\mathcal{H}} w_h, \tag{1}$$

subject to the following constraints.

TABLE IV
NOTATIONS OF THE ILP FORMULATIONS

Notation	Description
w_h	$= 1$ if a hypervisor is placed at potential hypervisor node $h \in \mathcal{H}$; 0, otherwise
$x_{h,s}$	$= 1$ if hypervisor node $h \in \mathcal{H}$ controls vSDN node $s \in \mathcal{S}$; 0, otherwise
$y_{h_1,h_2,s}$	$= 1$ if vSDN node $s \in \mathcal{S}$ is controlled by hypervisors $h_1, h_2 \in \mathcal{H}$; 0, otherwise
k	number of hypervisors in the optimal solution (i.e., $\|\mathcal{H}^*\|$)
a_r	$= 1$ if $r \in \mathcal{R}$ is acceptable; 0, otherwise
$a_{r,c}$	$= 1$ if $c \in \mathcal{C}$ can control every vSDN node $s \in \mathcal{S}_r$; 0, otherwise
$z_{c,s}$	$= 1$ if $c \in \mathcal{C}$ can control vSDN node $s \in \mathcal{S}$ through its currently active hypervisor pair

1) Hypervisor Activation Constraints: Eq. (2) ensures that only active hypervisors can control switches:

$$x_{h,s} \leq w_h \quad \forall s \in \mathcal{S}, \forall h \in \mathcal{H}, \tag{2}$$

while Eq. (3) says that hypervisors without controlled switches are inactive:

$$w_h \leq \sum_{s\in\mathcal{S}} x_{h,s} \quad \forall h \in \mathcal{H}. \tag{3}$$

2) Resilient Control Constraints: Each switch is controlled by a pair of hypervisors, except when there is a hypervisor at the switch's location when it controls itself (Case (2) in Definition 4):

$$x_{s,s} + \sum_{h\in\mathcal{H}} x_{h,s} = 2 \quad \forall s \in \mathcal{S}. \tag{4}$$

3) Valid Hypervisor-Pair Constraints: The hypervisor pair $\{h_1, h_2\}$ controls switch s if and only if both of them are controlling it, formulated in Eq. (5)-(7):

$$y_{h_1,h_2,s} \leq x_{h_1,s} \forall h_1, h_2 \in \mathcal{H}, \forall s \in \mathcal{S}, \tag{5}$$

$$y_{h_1,h_2,s} \leq x_{h_2,s} \forall h_1, h_2 \in \mathcal{H}, \forall s \in \mathcal{S}, \tag{6}$$

$$x_{h_1,s} + x_{h_2,s} - 1 \leq y_{h_1,h_2,s} \quad \forall h_1, h_2 \in \mathcal{H}, \forall s \in \mathcal{S}. \tag{7}$$

The hypervisor pairs which can control s in a latency-aware pathwise-disjoint fashion can be obtained from $\mathcal{T}$ in Eq. (8):

$$\sum_{(h_1,h_2,s)\in\mathcal{T}(s)} y_{h_1,h_2,s} = 1 \quad \forall s \in \mathcal{S}. \tag{8}$$

The above formulation gives a set of hypervisors $\mathcal{H}^*$ which has a minimum size $k = |\mathcal{H}^*|$. We will use this k number in Section V as the input hypervisor number for the full-knowledge algorithms to demonstrate the cost of our intelligent algorithm design which does not have any a priori knowledge about the vSDN requests.

V. Optimal Hypervisor Placement With Maximum Acceptance Ratio

In this section, we introduce an algorithm that provides the optimal locations of k hypervisors (obtained from Section IV-D, running the ILP_k algorithm) to maximize the number of accepted requests from a given vSDN request

set $\mathcal{R}$. We formulate the static LHPP problem in Section V-A, while Section V-B introduces our ILP formulation (denoted as ILP_a). We argue that the presented formulation is multi-purpose depending on the input set $\mathcal{R}$. On the one hand, if set $\mathcal{R}$ contains all currently embedded and future (arriving in the next time step) requests in the network, then the ILP has full knowledge to present the optimal hypervisor placement and gives us an optimal acceptance ratio we can use for comparisons. On the other hand, $\mathcal{R}$ can be a *representative set* of vSDN requests we expect – but has no exact knowledge about – in the future, which can be used either in LHPP for an initial hypervisor placement or in the static algorithm for planning hypervisor migrations. The usage and generation of representative sets will be discussed in Section V-C.

A. Static LHPP Problem Formulation

Similarly to other approaches [10], [15], [31] with full knowledge about the vSDN request set $\mathcal{R}$, our objective in the static problem formulation is to provide the resilient control architecture presented in Figure 1 (i.e., disjoint control path-pair satisfying the latency requirement traversing different hypervisor instances) to as many vSDN requests as possible, i.e., maximize the number of accepted requests. Using the pre-calculated $\mathcal{Q}$ set, a request $r \in \mathcal{R}$ is acceptable for a given hypervisor placement if

$$\exists c \in \mathcal{C} : \forall v^r \in \mathcal{V}^r \quad (c, h_1, h_2, v^r) \in \mathcal{Q},$$

where $\{h_1, h_2\}$ is the hypervisor pair controlling v^r. We define *acceptance ratio* as the fraction of acceptable requests in the request set:

$$a = \frac{\sum_{r \in \mathcal{R}} a_r}{|\mathcal{R}|}, \tag{9}$$

which will be used as the main performance metric. Therefore, the static LHPP problem can be formulated as follows:

Problem 2 [Static Latency-Aware Resilient Hypervisor Placement Problem (SHPP)]: Given $\mathcal{G}(\mathcal{V}, \mathcal{E})$, sets $\mathcal{S} \subseteq \mathcal{V}, \mathcal{H} \subseteq \mathcal{V}, \mathcal{C} \subseteq \mathcal{V}$, maximum global control path latency L and vSDN request set $\mathcal{R}$, find a latency-aware pathwise-disjoint control cover $\mathcal{H}' \subseteq \mathcal{H}$ for $\mathcal{S}$ with k hypervisors where the acceptance ratio of $\mathcal{R}$ is maximal.

B. Integer Linear Program for SHPP

Here, we present the mixed integer linear program of SHPP for a given request set $\mathcal{R}$. Note that in the experimental results for comparison, this ILP is used in two manners:

- If we know exactly all the current and future requests, the results are denoted as OPT.
- If we do not have exact information about the arriving requests, and we prepare our network for the current request set or for some representative set. The results are denoted as ILP_a.

Table IV specifies the parameters and the binary decision variables used in the formulation.

Similarly as in the ILP_k, variable w_h determines whether a hypervisor is located at the network node $h \in \mathcal{H}$ or not. Note that, after having solved the model, the variables w_h specify the set $\mathcal{H}^*$ of hypervisor nodes, specifically, $\mathcal{H}^* = \{h \in \mathcal{H} : w_h = 1\}$. The variable $x_{h,s}$ is set to one if switch $s \in \mathcal{S}$ is controlled by the hypervisor instance placed at node $h \in \mathcal{H}$. Similarly, the variable $y_{h_1,h_2,s}$ is set to one if switch $s \in \mathcal{S}$ is controlled by the hypervisor instances placed at nodes $h_1, h_2 \in \mathcal{H}$. Additionally to LHPP, in the SHPP ILP we need to add $\forall r \in \mathcal{R}$ request specific varaibles as well. The variable a_r is set to one if $r \in \mathcal{R}$ is acceptable, while $a_{r,c}$ indicates whether $c \in \mathcal{C}$ can control every vSDN node $v_r \in \mathcal{V}_r$. The variable $z_{c,s}$ is set to one if $c \in \mathcal{C}$ can control switch $s \in \mathcal{S}$ through a given hypervisor-pair. Finally, k represents the number of hypervisors which has to be used in the solution.

Our objective is to maximize the number of acceptable requests (or equivalently acceptance ratio) for $\mathcal{R}$:

$$max \sum_{r \in \mathcal{R}} a_r, \tag{10}$$

subject to the following constraints.

In order to obtain the resilient control architecture presented in Figure 1, we need the constraint groups 1)-3) from Section IV-D. Additionally, we need the following constraints for SHPP:

4) Hypervisor Number: There should be exactly k hypervisors in the solution:

$$\sum_{h \in \mathcal{H}} w_h = k. \tag{11}$$

5) Controller Competency Contraint: We obtain from quartets $\mathcal{Q}$ the possible hypervisor pairs $\{h_1, h_2\}$ that allow c to control s:

$$y_{h_1,h_2,s} \leq z_{c,s} \quad \forall (c, h_1, h_2, s) \in \mathcal{Q}. \tag{12}$$

If none of the hypervisor-pairs provide a latency-aware pathwise-disjoint control cover for the switch s, then the controller location c is not appropriate for s:

$$z_{c,s} \leq \sum_{(c,h_1,h_2,s) \in \mathcal{Q}} y_{h_1,h_2,s} \quad \forall s \in \mathcal{S}, \forall c \in \mathcal{C}. \tag{13}$$

6) Request Acceptability Contraints: A request can be accepted with a given controller location if it can control all switches in the request:

$$a_{r,c} \leq z_{c,s} \quad \forall s \in \mathcal{S}_r, \forall c \in \mathcal{C}, \forall r \in \mathcal{R}, \tag{14}$$

$$\sum_{s \in \mathcal{S}_r} z_{c,s} - |\mathcal{S}_r| + 1 \leq a_{r,c} \quad \forall c \in \mathcal{C}, \forall r \in \mathcal{R}. \tag{15}$$

The request is acceptable if there is a controller that can control all of its switches:

$$a_{r,c} \leq a_r \quad \forall c \in \mathcal{C}, \forall r \in \mathcal{R}, \tag{16}$$

while the request is not acceptable if there is no such controller:

$$a_r \leq \sum_{c \in \mathcal{C}} a_{r,c} \quad \forall r \in \mathcal{R}. \tag{17}$$

We primarily use the above formulation to calculate an optimal solution (i.e., with maximum acceptance ratio) for a certain request set $\mathcal{R}$. Note that, owing to its high complexity,

TABLE V
NETWORK CHARACTERISTICS ($|V|$, $|E|$, γ – AVERAGE NODE DEGREE)

Network	$\|V\|$	$\|E\|$	γ
Janos-US [43]	26	42	3.23
Italy [44]	25	35	2.72
COST 266 [43]	37	57	3.08
Germany [43]	50	88	3.52

the possible huge number of requests in $\mathcal{R}$, and the frequent usage in each time step, this static algorithm has little practical relevance in a dynamically changing environment. Thus, we only use it as an optimal value to demonstrate the efficiency of our intelligent LHPP algorithms. However, in Section V-C, we show that the same ILP formulation can be used in LHPP to distinguish the quality of different hypervisor placements using a representative request set $\mathcal{R}_{rep}$, obtained either purely from graph metrics or historical data where available.

C. Representative Request Set Generation

It is obvious that if an empty set is used as the representative set, i.e., $\mathcal{R}_{rep} = \emptyset$, the ILP for SHPP (denoted as ILP_a) is the same as the ILP for LHPP (denoted as ILP_k) and the solution is one of the many possible latency-aware hypervisor placements. These possible solutions can provide very different acceptance ratios for the possible request sets: some of them are generally dysfunctional placements and have low acceptance ratio for most request sets and high for only a few, while some of them are generally good placements and have high acceptance ratio for most request sets and low for only a few cases. The representative request set aims to separate the wheat from the chaff: it filters out the dysfunctional hypervisor placements from the possible solutions, leaving mainly "good" hypervisor placements as possible solutions. In this way, $\mathcal{R}_{rep}$ helps us prepare our hypervisor placement for the future.

Since the future requests are unspecified, it is impossible to construct a representative request set that perfectly resembles them. If the representative set is not selective enough (too small, contains mostly easily acceptable or unacceptable requests), it cannot filter out the "inferior" hypervisor placements. On the other hand, if the representative set is not broad/universal enough (i.e., it contains mostly alike requests), it will favor some hypervisor placements that are not necessarily good overall. Note that ILP_a gets most of its information from the acceptable requests in $\mathcal{R}_{rep}$; thus, an overall good representative set should contain mostly acceptable requests, but not too easily acceptable ones. We will show that with a carefully selected small representative request set (≈0.1% of all possible vSDN requests), the optimal solution can be approached with a significantly lower runtime, presented in Section VI.

VI. EXPERIMENTAL RESULTS

In this section, we provide a comprehensive study of the different aspects of our solutions. In particular, first, in Section VI-A we analyze the preprocessing steps and the quartets. In Section VI-B we present the request generation methods utilized for the experimental results, meanwhile in Section VI-C the guidelines for the representative set generation are discussed. Finally, in Section VI-D the preparedness analysis is presented.

TABLE VI
RUNTIME OF SHORTEST PATH PRE-CALCULATION, $\mathcal{Q}$ GENERATION AND NUMBER OF QUARTETS $|\mathcal{Q}|$ IN THE 25-NODE ITALY NETWORK ($L = 1.0$)

P	$\mathcal{P}$ calculation [s]	$\mathcal{Q}$ generation [s]	$\|\mathcal{Q}\|$
1	0.08	1.31	36472
2	0.24	2.2	54941
4	0.65	4.11	65988
8	1.62	9.7	68712
16	3.73	33.75	69150
32	8.17	146.24	69194
64	18.42	499.53	69194

The investigated real-life optical backbone networks are given in Table V. The Janos-US, COST 266, and Germany network topologies are obtained from [43], and the link lengths are defined as the distances between nodes assuming that the node locations (given with latitude and longitude coordinates) are known for each node. For Italy [44], the links are defined as a series of points (starting with the source and ending with the target node) with straight lines between them. Therefore, the link length is defined as the summed length of the straight lines of the link assuming that longitude and latitude coordinates are known for each point. In our experiments, without loss of generality, we investigate the most general version of Problem 1 where $\mathcal{S} = \mathcal{H} = \mathcal{C} = \mathcal{V}$. The global latency requirement is given as a ratio to the diameter of the network, i.e., $L = 0.5$ means that the latency requirement is 0.5 times the diameter of the network.

We conduct our simulations on a virtual machine with 8 cores (Intel Xeon E5-2630 v3 @ 2.4GHz) and 32GB of RAM running Ubuntu 18.04.1 LTS with kernel 4.15.0-151-generic. The simulation environment and the algorithms are implemented in Python 3.8.2. The ILP instances are created with the Gurobi Python Interface (gurobipy) and solved with the Gurobi solver (version 9.1.2) [45].

A. Preprocessing Analysis

This subsection investigates the complexity of the preprocessing steps and provides an analysis of the quartets. In the preprocessing phase, *we calculate the P-shortest simple paths between each node-pair* and generate $\mathcal{Q}$ and $\mathcal{T}$. We showed in Lemma 1 that the pre-calculation times depend on the network topology and the P parameter. Similarly, the number of quartets ($|\mathcal{Q}|$) depends on the network topology, the P parameter, and the global latency requirement L. If P is too low, some of the possible quartets cannot be found, resulting in lower performance. If P is too high, the pre-calculation times are unnecessarily high. Table VI presents the runtimes of the path calculation and quartet generation for several P values. The size of $\mathcal{Q}$ varies only little for $P \geq 16$ but the runtime of the preprocessing grows significantly. According to these results, we select $P = 16$ for further investigations since it provides a great balance between runtime and accuracy.

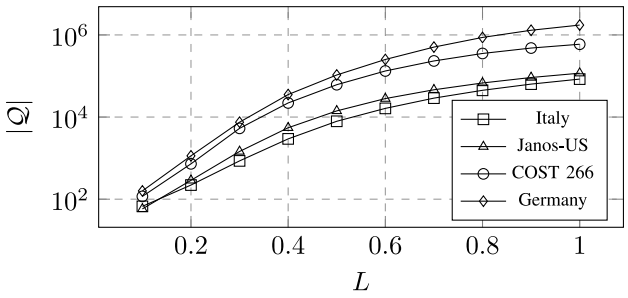


Fig. 3. Number of quartets at different global latency requirements in the analysed networks.

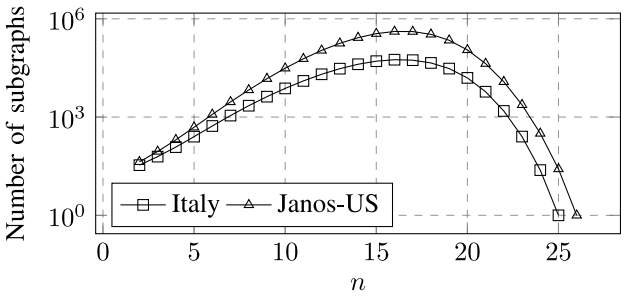


Fig. 4. The number of possible subgraphs of size n in the Italy and Janos-US networks.

Figure 3 shows the number of quartets for all the investigated networks subject to the global latency requirement. At low L values, there are only a few thousand of quartets, but their number increases drastically with L even reaching 10^6 in case of the Germany network. Note that while the Germany network has twice more nodes and edges than the Janos-US network it has nearly 10 times more quartets.

B. vSDN Request Generation

The vSDN requests are connected subgraphs in our network (cf. Section III-B). Therefore, to generate requests, we use the *Simple* method from [46] to enumerate all connected induced subgraphs of size n. It has a complexity of $O(n^2 \times \delta_{max})$ where δ_{max} is the maximal node degree in the graph. The authors provide an implementation of their algorithms in Python that we use in our framework.

Let $\mathcal{R}_n$ and $\mathcal{R}_{\leq n}$ denote the set of all connected induced subgraphs of size n and $\leq n$ in the network, respectively. Therefore, $\mathcal{R}_{\leq n} = \bigcup_{i=2}^{n} \mathcal{R}_n$. It is apparent that set $\mathcal{R}_{\leq |\mathcal{V}|}$ contains all possible subgraphs of the network while set $\mathcal{R}_2$ contains all adjacent node pairs. Figure 4 presents the number of subgraphs according to the size of the subgraphs for the Italy and Janos-US networks.

The $\mathcal{R}_n$ sets are used for benchmarking purposes as it helps to estimate the performance of the hypervisor placement on requests with a given size, which are selected from $\mathcal{R}_{|\mathcal{V}^r|}$, where $|\mathcal{V}^r|$ is the fixed size of the requests. The $\mathcal{R}_{\leq n}$ sets represent real-life-like requests with given maximal size, and they are used to generate the representative set for ILP_a in our evaluations. In this method, a given number of subgraphs are selected from $\mathcal{R}_{\leq |\mathcal{V}^r|_{max}}$ where $|\mathcal{V}^r|_{max}$ is the maximal request size. The vSDN request count is an upper bound for the number of generated subgraphs since in some cases it is greater than the number of possible subgraphs.

C. Representative Request Set Generation

In this subsection, we study the effect of the representative request set (denoted as $\mathcal{R}_{rep}$) generation on the acceptance ratio of ILP_a, which corresponds to the preparedness of the hypervisor placement. For demonstration purposes, we present the results of the Italy network, but the trends and the take-away message is general to all the topologies studied. In our simulations, $\mathcal{R}_{rep}$ is sampled from $\mathcal{R}_{\leq n}$, where the maximal request size n varies between 2 and $|\mathcal{V}|/2$ and the number of requests varies between 2 and 200. For each representative set parameter combination, a preparedness analysis is performed with the same settings as in Section VI-D. Table VII contains the results for given representative set parameters.

TABLE VII
AVERAGE ACCEPTANCE RATIO OF THE ILP_a WITH DIFFERENT $\mathcal{R}_{rep}$ PARAMETERS IN THE ITALY NETWORK (AVERAGES BASED ON 10 INDEPENDENT RUN)

$\lvert\mathcal{V}^r\rvert_{max}$	$\lvert\mathcal{R}_{rep}\rvert$						
	2	5	10	20	50	100	200
2	0.40	0.42	0.52	0.49	0.44	0.53	0.40
3	0.37	0.50	0.45	0.60	0.51	0.60	0.60
4	0.45	0.54	0.55	0.61	0.62	0.62	0.63
5	0.44	0.56	0.59	0.61	0.64	0.64	0.64
6	0.45	0.57	0.61	0.61	0.63	0.64	0.64
7	0.47	0.56	0.59	0.60	0.64	0.64	0.64
13	0.57	0.58	0.59	0.61	0.62	0.63	0.64

To compare the different parameter combinations, the average acceptance ratio is calculated over all 10 runs and vSDN request sizes ranging from 2 to $|\mathcal{V}|/2$. As expected, when $\mathcal{R}_{rep}$ contains easily acceptable requests, e.g., $|\mathcal{V}^r|_{max} = 2$, ILP_a achieves lower acceptance ratio. Similarly, from a small representative set ($|\mathcal{R}_{rep}| \leq 10$) less information can be obtained by ILP_a resulting in a lower acceptance ratio again. According to the simulations, ILP_a makes the most out of challenging representative sets ($|\mathcal{R}_{rep}| \geq 50$ and $|\mathcal{V}^r|_{max} \geq 4$) achieving acceptance ratios consistently over 0.62, above these boundaries no parameter combination seems superior to the others. The same tendencies can be observed in the other networks. Since we assume that no specific knowledge is available about the future requests and corresponding vSDN embeddings, we use $|\mathcal{R}| = 100$ and $|\mathcal{V}^r|_{max} = \frac{|\mathcal{V}|}{4}$ in our simulations.

D. Preparedness Analysis

In this subsection, we evaluate the hypervisor placement methods on the presented network topologies (Table V) in a static scenario. The analysis focuses on the number of hypervisors used, hypervisor usage probability, and acceptance ratio of the requests. The evaluation settings are summarized in Table VIII. For each setting, the hypervisor placement and the evaluation are repeated 10 times with different seeds. Each evaluation consists of multiple vSDN request sets with specific request sizes (from 2 to $|\mathcal{V}|$), each containing 100 requests if possible. Note that the hypervisor placements are evaluated on the same request sets in the corresponding runs (e.g., 1st run).

TABLE VIII
EVALUATION SETTINGS IN THE PREPAREDNESS ANALYSIS

Parameter	Values
Network topology	Italy, Janos-US COST 266, Germany
Latency requirement	$0.1, \ldots, 1.0$
P	16
vSDN request size ($\lvert\mathcal{V}^r\rvert$)	$2, \ldots, \lvert\mathcal{V}\rvert$
vSDN request count	100 (maximal)
$\lvert\mathcal{R}_{rep}\rvert$	100
max request size in $\mathcal{R}_{rep}$	$\frac{\lvert\mathcal{V}\rvert}{4}$
Hypervisor Placement	Alg. 1, ILP$_k$, ILP$_a$, OPT
Runs per setup	10

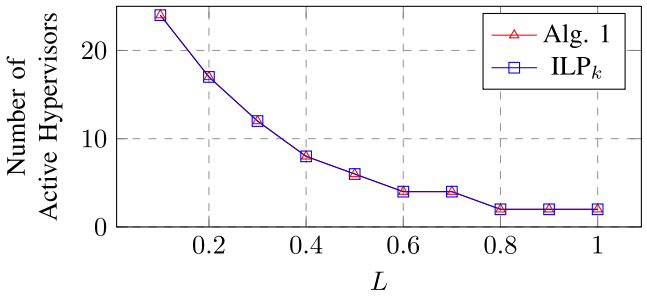


Fig. 5. The number of active hypervisors subject to the global latency requirement L in the Italy network.

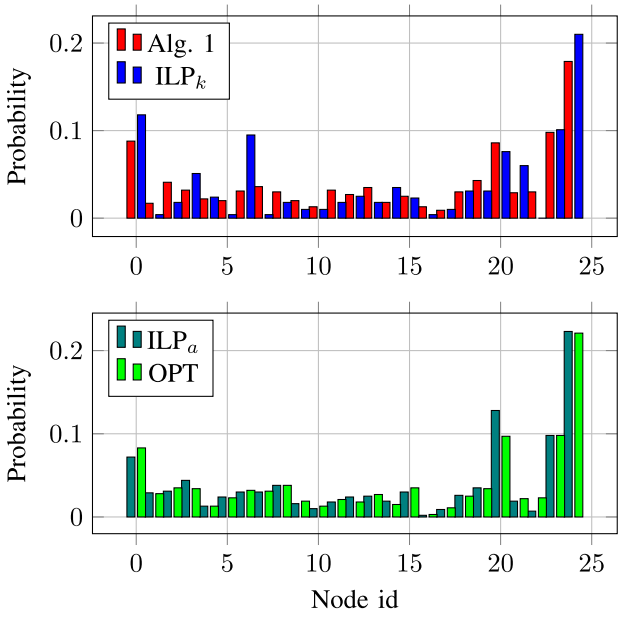


Fig. 6. Probability of the used hypervisor locations of the placement methods in the Italy network based on the 10 simulation runs.

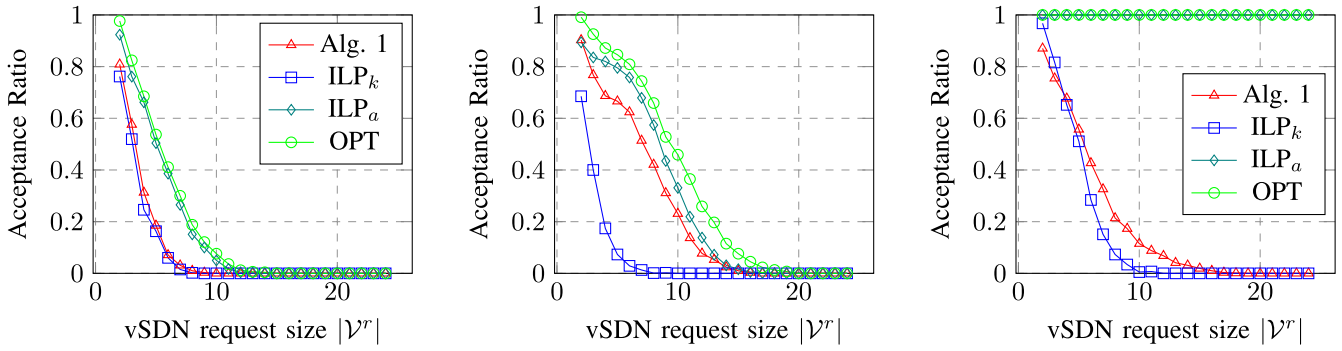
The hypervisor placement methods of the LHPP, i.e., Algorithm 1 and ILP$_k$, are calculated solely based on the network topology, the path count parameter P, and the global latency requirement L. They do not take into account any knowledge about the possible requests, and their goal is to find a placement with a minimal number of hypervisors. On the other hand, ILP$_a$ uses a representative request set $\mathcal{R}_{rep}$ to obtain a more suitable placement than ILP$_k$ generated according to the findings in Section VI-C. Additionally, to assess how close our placement methods are to the theoretical maximum in terms of acceptance ratio, OPT is used to find the hypervisor placement with the highest acceptance ratio for each evaluated request set. Note that while the other placements are calculated once in each run and evaluated for many request sets (with different request sizes) OPT is calculated for each request set in each run resulting in the theoretical maximum acceptance ratio for each request set that may not be reachable in practice.

Algorithm 1 and ILP$_k$ are the two methods that try to find a placement with a minimal number of hypervisors. Figure 5 presents the results for the Italy network which show that both algorithms deliver the same minimal number of hypervisors in each case. Clearly, at lower L values, more hypervisors are required to resiliently control the switches, while at higher L values, less is sufficient. For example, at $L = 0.4$, eight hypervisors must be used in the Italy network, but at $L = 0.6$, four are sufficient.

Despite the consensus on the number of hypervisors, the algorithms differ in the hypervisor locations. To illustrate this, Figure 6 presents the usage frequency of hypervisor locations. It aggregates results from all configurations listed in Table VIII. The methods optimizing for maximum acceptance ratio, i.e., ILP$_a$ and OPT agree mostly with Algorithm 1 and ILP$_k$ on the hypervisor locations apart from some minor differences. Not surprisingly, the most frequent hypervisor locations are all central nodes in the network (e.g., nodes 0, 20, 23, and 24 of the Italy network are in Central-Italy). Although the hypervisor usage of Algorithm 1 and ILP$_k$ seems not too distinct from the hypervisor usage of ILP$_a$ and OPT, the minor differences significantly impact the acceptance ratio.

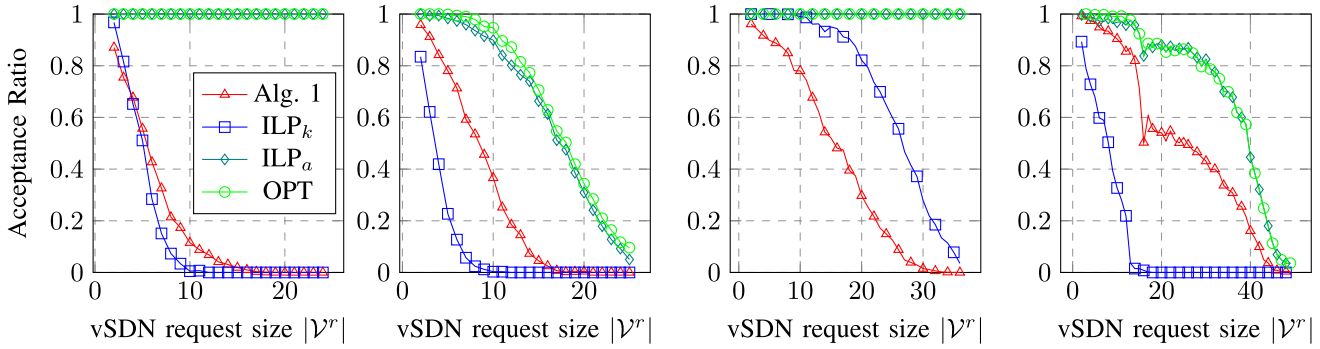
Figure 7 shows the average acceptance ratio of the hypervisor placement methods for several vSDN request sizes at three global latency requirements. Next to each L value the number of active hypervisors is also displayed. In Figure 7(a), the low L is very strict and leaves little room for hypervisor placement optimization thus only the small vSDN requests ($\lvert\mathcal{V}^r\rvert \leq 5$) have an acceptance ratio over 50%. In Figure 7(b), L is large enough to enable higher acceptance ratios. Note that the occasional significant gap (such as in Figure 7(b)) between Algorithm 1 and ILP$_k$ is due to the ability of Algorithm 1 to randomly select a solution from many (i.e., 400) in the end, while ILP$_k$ returns the first feasible solution, which usually performs poorly according to the simulations. At $L = 0.6$, it is possible to accept every request with a carefully selected hypervisor placement which is exactly what ILP$_a$ does. The large gap between the acceptance ratios of Figure 7(b) and 7(c) is caused by the relaxation of the latency requirement (from 0.5 to 0.6) and the decrease of active hypervisors (from 6 to 4).

One can observe that ILP$_a$ with the proposed representative set generation outperforms both Algorithm 1 and ILP$_k$, and finds the placement with the best acceptance ratio among the three approaches, always close to the optimal. Figure 8 presents the acceptance ratio of the hypervisor placement methods subject to the vSDN request size for all four networks with $L = 0.6$. The results of each network show the impressive benefits of the representative request set as ILP$_a$ consistently performs much better than Algorithm 1 and ILP$_k$. In the case



(a) $L = 0.4$ $(k = 8)$ (b) $L = 0.5$ $(k = 6)$ (c) $L = 0.6$ $(k = 4)$

Fig. 7. Comparison about the acceptance ratio of the hypervisor placement methods in the Italy network subject to the vSDN request size ($|\mathcal{V}^r|$) for given latency requirement L (averages based on 10 independent runs). After each L value the number of active hypervisors (k) is displayed.



(a) Italy $(k = 4)$ (b) Janos-US $(k = 4)$ (c) COST 266 $(k = 3)$ (d) Germany $(k = 3)$

Fig. 8. Comparison about the acceptance ratio of the hypervisor placement methods in the investigated networks subject to the vSDN request size ($|\mathcal{V}^r|$) for latency requirement $L = 0.6$ (averages based on 10 independent runs). After each network name the number of active hypervisors (k) is displayed.

of the Italy and COST 266 networks (Figure 8(a) and 8(c)), ILP_a is able to find the optimal solution, while in the case of the Janos-US and Germany networks, it finds a solution that is very close to the optimal one. On the COST 266 network with $L = 0.6$ (Figure 8(c)), ILP_k performs surprisingly well, much better than Algorithm 1. In this case, the acceptance ratio of the first feasible solution is much better than the average of the random ones. In summary, we can state that the representative requests enable us to reliably obtain a hypervisor placement close to the optimal in terms of acceptance ratio.

VII. Conclusion

In this paper we introduced a novel latency-aware control plane design with primary/backup hypervisors and edge-disjoint control paths to protect against single-link and hypervisor failures. Based on this design, the latency-aware resilient hypervisor placement problem was formulated. We proved that minimizing the number of hypervisors is NP-hard and hard to approximate, and proposed a polynomial-time algorithm based on greedy set cover and an integer linear programming model. Our simulation results show that our algorithm approaches the optimal solution in terms of hypervisor number. Furthermore, we extended the previous problem formulation by adding a request set for which the hypervisor placement providing the highest acceptance ratio must be determined. We presented an integer linear programming formulation for this problem and argued that it can be used not only to find the optimal placement for a certain request set but also to obtain a hypervisor placement prepared for the future by utilizing a representative request set. For evaluation of the three hypervisor placement methods, we have investigated the impact of the hypervisor placement on the vSDN request acceptance ratio. Our results show that the representative set significantly improves the hypervisor placement and helps to obtain close-to-optimal acceptance ratio in every scenario.

As a future work, we will analyse our proposed placement methods in a dynamic scenario, where the network needs to migrate hypervisors in order to meet the SLA requirements of the currently active vSDNs and maximize the acceptance ratio of the newly arrived requests.

References

[1] H. Viswanathan and P. E. Mogensen, "Communications in the 6G era," *IEEE Access*, vol. 8, pp. 57063–57074, 2020.

[2] A. Shahraki, M. Abbasi, M. J. Piran, M. Chen, and S. Cui, "A comprehensive survey on 6G networks: Applications, core services, enabling technologies, and future challenges," 2021, *arXiv:2101.12475*.

[3] Y. Lu and X. Zheng, "6G: A survey on technologies, scenarios, challenges, and the related issues," *J. Ind. Inf. Integr.*, vol. 19, Sep. 2020, Art. no. 100158.

[4] "6G—Connecting a Cyber-Physical World." [Online]. Available: https://www.ericsson.com/en/reports-and-papers/white-papers/a-research-outlook-towards-6g (Accessed: Apr. 30, 2022).
[5] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6G networks: Use cases and technologies," *IEEE Commun. Mag.*, vol. 58, no. 3, pp. 55–61, Mar. 2020.
[6] C. J. Bernardos and M. A. Uusitalo. "European Vision for the 6G Network Ecosystem." 2021. [Online]. Available: https://zenodo.org/record/5007671
[7] L. U. Khan, I. Yaqoob, N. H. Tran, Z. Han, and C. S. Hong, "Network slicing: Recent advances, taxonomy, requirements, and open research challenges," *IEEE Access*, vol. 8, pp. 36009–36028, 2020.
[8] 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.
[9] A. Blenk, A. Basta, W. Kellerer, and S. Schmid, "On the impact of the network hypervisor on virtual network performance," in *Proc. IFIP Netw. Conf. (IFIP Netw.)*, 2019, pp. 1–9.
[10] B. P. R. Killi and S. V. Rao, "On placement of hypervisors and controllers in virtualized software defined network," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 2, pp. 840–853, Jun. 2018.
[11] D. Basu, A. Jain, U. Ghosh, and R. Datta, "QoS-aware controller and hypervisor placement in vSDN-enabled 5G networks for time-critical applications," in *Proc. IEEE INFOCOM WKSHPS*, 2021, pp. 1–6.
[12] S. Amjad, A. Varasteh, N. Deric, and C. Mas-Machuca, "Delay-aware dynamic hypervisor placement and reconfiguration in virtualized SDN," in *Proc. 12th Int. Conf. Netw. Future (NoF)*, 2021, pp. 1–9.
[13] P. Babarczi, "Resilient control plane design for virtual software defined networks," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 2557–2569, Sep. 2021.
[14] M. Tornatore *et al.*, "Alert-based network reconfiguration and data evacuation," in *Guide to Disaster-resilient Communication Networks*, J. Rak and D. Hutchinson, Eds. Cham, Switzerland: Springer, 2020, ch. 14, pp. 1–24.
[15] A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer, "Control plane latency with SDN network hypervisors: The cost of virtualization," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 366–380, Sep. 2016.
[16] S. Chen, W. Sun, and W. Hu, "On dynamic hypervisor placement in virtualized software defined networks (vSDNs)," in *Proc. 22nd Int. Conf. Transparent Opt. Netw. (ICTON)*, 2020, pp. 1–5.
[17] O. Dobrijevic, C. Natalino, M. Furdek, H. Hodzic, M. Dzanko, and L. Wosinska, "Another price to pay: An availability analysis for SDN virtualization with network hypervisors," in *Proc. 10th Int. Workshop Resilient Netw. Des. Model. (RNDM)*, 2018, pp. 1–7.
[18] L. Schiff, S. Schmid, and M. Canini, "Ground control to major faults: Towards a fault tolerant and adaptive SDN control network," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. Workshop (DSN-W)*, 2016, pp. 90–96.
[19] M. Canini, I. Salem, L. Schiff, E. M. Schiller, and S. Schmid, "Renaissance: A self-stabilizing distributed SDN control plane," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2018, pp. 233–243.
[20] A. Blenk, A. Basta, and W. Kellerer, "HyperFlex: An SDN virtualization architecture with flexible hypervisor function allocation," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM)*, 2015, pp. 397–405.
[21] D. Bertsimas, D. B. Brown, and C. Caramanis, "Theory and applications of robust optimization," *SIAM Rev.*, vol. 53, no. 3, pp. 464–501, 2011.
[22] A. D. Wissner-Gross and C. E. Freer, "Causal entropic forces," *Phys. Rev. Lett.*, vol. 110, Apr. 2013, Art. no. 168702.
[23] K. Kar, M. Kodialam, and T. V. Lakshman, "Minimum interference routing of bandwidth guaranteed tunnels with MPLS traffic engineering applications," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 12, pp. 2566–2579, Dec. 2000.
[24] C. Griner, J. Zerwas, A. Blenk, S. Schmid, M. Ghobadi, and C. Avin, "Cerberus: The power of choices in datacenter topology design—A throughput perspective," in *Proc. ACM SIGMETRICS*, 2021, pp. 1–33.
[25] A. S. Klyubin, D. Polani, and C. L. Nehaniv, "Empowerment: A universal agent-centric measure of control," in *Proc. IEEE Congr. Evol. Comput.*, vol. 1, Sep. 2005, pp. 128–135.
[26] P. Babarczi *et al.*, "A mathematical framework for measuring network flexibility," *Comput. Commun.*, vol. 164, pp. 13–24, Dec. 2020.
[27] A. Blenk, "Towards virtualization of software-defined networks: Analysis, modeling, and optimization," Ph.D. dissertation, Dept. Elect. Comput. Eng., Technische Universität München, Munich, Germany, 2018.
[28] P. Kalmbach, J. Zerwas, P. Babarczi, A. Blenk, W. Kellerer, and S. Schmid, "Empowering self-driving networks," in *Proc. ACM SIGCOMM Workshop Self Driving Netw. (SelfDN)*, Aug. 2018, pp. 8–14.
[29] P. Vizarreta, C. M. Machuca, and W. Kellerer, "Controller placement strategies for a resilient SDN control plane," in *Proc. Workshop Resilient Netw. Des. Model. (RNDM)*, Sep. 2016, pp. 253–259.
[30] D. S. Johnson *et al.*, "Near-optimal disjoint-path facility location through set cover by pairs," *Oper. Res.*, vol. 68, no. 3, pp. 896–926, 2020.
[31] A. Blenk, A. Basta, J. Zerwas, and W. Kellerer, "Pairing SDN with network virtualization: The network hypervisor placement problem," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, 2015, pp. 198–204.
[32] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Fast failure recovery for in-band OpenFlow networks," in *Proc. IEEE Des. Rel. Commun. Netw. (DRCN)*, 2013, pp. 52–59.
[33] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "In-band control, queuing, and failure recovery functionalities for OpenFlow," *IEEE Netw.*, vol. 30, no. 1, pp. 106–112, Jan./Feb. 2016.
[34] R. Khalili, Z. Despotovic, and A. Hecker, "Flow setup latency in SDN networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2631–2639, Dec. 2018.
[35] Y. Zhang, N. Beheshti, and M. Tatipamula, "On resilience of split-architecture networks," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, 2011, pp. 1–6.
[36] B. P. R. Killi and S. V. Rao, "Optimal model for failure foresight capacitated controller placement in software-defined networks," *IEEE Commun. Lett.*, vol. 20, no. 6, pp. 1108–1111, Jun. 2016.
[37] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in SDN-based core networks," in *Proc. Teletraffic Congr. (ITC)*, 2013, pp. 1–9.
[38] R. Hassin and D. Segev, "The set cover with pairs problem," in *Proc. Int. Conf. Found. Softw. Technol. Theor. Comput. Sci.*, 2005, pp. 164–176.
[39] R. Raz and S. Safra, "A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP," in *Proc. ACM Symp. Theory Comput.*, 1997, pp. 475–484.
[40] J. Y. Yen, "Finding the k shortest loopless paths in a network," *Manag. Sci.*, vol. 17, no. 11, pp. 712–716, 1971.
[41] L. Lovász, "On the ratio of optimal integral and fractional covers," *Discrete Math.*, vol. 13, no. 4, pp. 383–390, 1975.
[42] D. S. Johnson, "Approximation algorithms for combinatorial problems," *J. Comput. Syst. Sci.*, vol. 9, no. 3, pp. 256–278, Dec. 1974.
[43] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0—Survivable network design library," in *Proc. INOC*, 2007, pp. 276–286.
[44] A. Valentini *et al.*, "Network resiliency against earthquakes," in *Proc. Workshop Resilient Netw. Des. Model. (RNDM)*, 2019, pp. 1–7.
[45] (Gurobi Optim., LLC, Beaverton, OR, USA). *Gurobi Optimizer Reference Manual*. (2021). [Online]. Available: https://www.gurobi.com
[46] C. Komusiewicz and F. Sommer, "Enumerating connected induced subgraphs: Improved delay and experimental comparison," *Discrete Appl. Math.*, vol. 303, pp. 262–282, Nov. 2021.

Ferenc Mogyorósi received the M.Sc. degree (*summa cum laude*) in electrical engineering from the Budapest University of Technology and Economics (BME), Hungary, in 2020. He is currently pursuing the Ph.D. degree with the High-Speed Networks Laboratory, Department of Telecommunications and Media Informatics, Doctoral School of Electrical Engineering, BME. He was awarded the UNKP Doctoral Fellowship of the Hungarian Ministry of Innovation and Technology (ITM) for the university year 2021–2022 and the Cooperative Doctoral Programme Doctoral Scholarship of the ITM. His research interests focus on survivability in optical backbone networks, artificial intelligence, and mobile positioning.

Péter Babarczi (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees (*summa cum laude*) in computer science from the Budapest University of Technology and Economics (BME), Hungary, in 2008 and 2012, respectively. From 2017 to 2019, he was an Alexander von Humboldt Postdoctoral Research Fellow with the Chair of Communication Networks, Technical University of Munich, Germany. He is currently working as an Associate Professor with the Department of Telecommunications and Media Informatics, BME. His current research interests include intelligent self-driving networks, multi-path Internet routing, cloud gaming, network coding in transport networks, and combinatorial optimization in softwarized networks. He received the János Bolyai Research Scholarship of the Hungarian Academy of Sciences in 2013, and the Postdoctoral Research Fellowship of the Alexander von Humboldt Foundation in 2017. Since 2020, he has been the Lead Researcher of an OTKA FK Young Researchers' Excellence Programme supported by the National Research, Development and Innovation Fund of Hungary.

Johannes Zerwas (Graduate Student Member, IEEE) received the M.Sc. degree in electrical engineering and information technology from the Technical University of Munich (TUM), Germany, in 2018. He joined the Chair of Communication Networks, TUM as a Research and Teaching Associate in February 2018. His research is focused on flexible and predictable network virtualization, adaptive topologies, as well as data-driven networking algorithms.

Andreas Blenk received the Dr.-Ing. degree (Ph.D.) (Highest Distinction) *(summa cum laude)* from the Technical University of Munich in 2018. He is currently a Research Scientist with Siemens AG, Munich, Germany, and a Lecturer with the Chair of Communication Networks, Technical University of Munich, Munich. His research interests are data-driven and machine learning-based network designs, algorithms, and network monitoring.

Alija Pašić received the M.Sc. (*summa cum laude*) and Ph.D. (*summa cum laude*) degrees in electrical engineering from the Budapest University of Technology and Economics, Hungary, in 2013 and 2019, respectively, where he is currently an Assistant Professor with the High-Speed Networks Laboratory, Department of Telecommunications and Media Informatics. His research interests focus on survivability in optical backbone networks, network coding, machine learning, artificial intelligence, and mobile positioning. He received the János Bolyai Research Scholarship of the Hungarian Academy of Sciences in 2021. Since 2021, he has been the Lead Researcher of an OTKA PD postdoctoral Excellence Programme supported by the National Research, Development and Innovation Fund of Hungary.