Resilient Control Plane Design for Virtual Software Defined Networks

Péter Babarczi, *Member, IEEE*

***Abstract*—Control plane survivability in virtual software-defined networks (vSDN) – where multiple tenants share the same physical infrastructure – is even more critical than in normal SDN networks. A reliable communication channel from the switches through the network hypervisor to the virtual controller is inevitable in order to avoid state inconsistencies, tenant isolation and security issues on the virtual switches. Although reliable controller placement and control plane design was thoroughly investigated in SDNs, there was a lack of attention for resilient hypervisor placement and control path design for vSDNs. Therefore, in this paper we make a two-fold contribution towards a survivable vSDN control plane. First, we propose an approximation algorithm for (hypervisor) placement which – in contrast with traditional approaches which minimize the average latency to the hypervisors as an objective function – focuses on finding the appropriate number of hypervisor instances to satisfy the control path length constraints declared in the service level agreements, leaving enough options open for self-driving network designs and intelligent algorithms. Second, we propose a general dynamic program that calculates minimum length paths traversing specific type of nodes in a given order, and apply it to find control paths from the virtual controller of the slice to the virtual switches traversing the corresponding hypervisor location. We conduct thorough simulations on real-world topologies to demonstrate the effectiveness of our approaches in no failure and single link failure scenarios.**

***Index Terms*—virtual networks, resilient hypervisor placement, facility location, function chain routing, intelligent algorithms**

I. Introduction

In the last decades communication networks transformed from a best-effort network connecting a handful of super-computers to a critical infrastructure connecting millions of devices. During this evolution the complexity of networks increased, and the management of nowadays Internet is a hard task for human operators and often leads to misconfiguration and service outages. However, network reliability is crucial both for mission-critical applications built on these networks (e.g., telesurgery) and for service providers to reach high Quality of Experience and hence, user satisfaction for their customers. These challenges fostered the concept of self-driving networks [1], which can react to unforeseen challenges in a timely manner without the need of human intervention.

The proliferation of Software Defined Networking (SDN) and network (function) virtualization eases the reconfiguration of networks to respond to challenges, thus, providing enough flexibility [2] and tool-set for self-driving networks to control themselves based on their own observations and analysis of the network states. Therefore, "intelligent" algorithms are required for such self-driving networks to steer themselves into stable states upon traffic fluctuations or provide enough resilience against network failures [1]. By intelligence in this context we mean that the intrinsic motivation of these algorithms should be to keep as many future options open as possible (e.g., do not reserve or create bottlenecks), which can be used to react to unforeseen challenges [3]. Hence, intelligent algorithms need to be general enough and cannot rely on finely tuned objective functions tweaked to the current parameters of the network.

In virtual Software-Defined Networks (vSDN) multiple tenants share the same physical SDN network [4]. Each tenant is provided by a given slice of network resources (e.g., forwarding table space at switches, link bandwidth, CPU cores, etc.), where they can configure their virtual switches with their own controller and operate their own virtual network. The network hypervisor layer (consists of a single or multiple hypervisor instances) ensures isolation (virtualization) of network attributes between different tenants, and also provides the abstraction of the reserved resources to the controller(s). In order to provide these functionalities, all control traffic of the slices between the switches and the controller goes through the hypervisor layer; therefore, even the failure of a single hypervisor instance could disrupt the traffic of several virtual networks. Although resilience of SDN transport networks against single link failures was thoroughly investigated [5], despite of its criticality hypervisor placement and control path resilience of vSDNs was barely considered. Hence, in this paper we will focus on this problem while we keep intelligent algorithm design and self-driving networks in mind.

It was shown [6]–[8] that the 50 ms recovery time requirement of carrier SDN networks can be achieved only if the switches do not have to contact the controller during the recovery process. Hence, protection paths have to be pre-computed and backup rules have to be deployed in the switches in advance of service disruptions both for data plane flows and in-band control plane paths. Thus, in this paper we will investigate the resilient hypervisor placement problem together with pre-calculating control paths for the virtual networks. Instead of optimizing for a given objective function and finding the best possible placement, we are looking for a "good enough" solution which satisfies the service level agreements (SLA) and might leave enough options open for future self-driving designs and intelligent algorithms. We believe that the additional freedom provided by our approach (i.e., selecting from multiple acceptable hypervisor locations rather than

The work of P. Babarczi was supported in part by Project no. 134604 that has been implemented with the support provided by the National Research, Development and Innovation Fund of Hungary, financed under the FK_20 funding scheme, and in part by the Alexander von Humboldt Foundation.

P. Babarczi is with the MTA-BME Future Internet Research Group, Dept. of Telecommunications and Media Informatics, Faculty of Electrical Engineering and Informatics (VIK), Budapest University of Technology and Economics (BME), Hungary (e-mail: babarczi@tmit.bme.hu).

having a single one as the result of a fine-tuned optimization objective) can serve as the building block of these future networks.

The rest of the paper is organized as follows. Section II summarizes the related work on resilient control plane design. We formulate our problem in Section III and propose an algorithm based on minimum set cover for resilient hypervisor placement in Section IV. After the hypervisor locations are fixed, we investigate the problem of finding the controller location of dynamically arriving virtual network requests and calculate the corresponding unprotected and resilient control paths with a general dynamic program proposed for arbitrary function chains in Section V. Finally, we present our simulation results in Section VI and conclude the paper in Section VII.

II. Related Work

A. Pro-Active Resilient Control Path Design

It was shown that the strict 50 ms failure recovery time requirement (RFC 5654) for the data plane is achievable only if the recovery paths are pre-computed and established together with the working paths [6]. Although implementing protection approaches for single link failure resilience [9], [10] requires extra entries in the switches (using the group table concept), it avoids the sudden increase of traffic load on the single controller instance upon failure detection and loss of data packets due to the lengthy recovery time of the restoration approaches. This concept [6] was extended to in-band control channel failure recovery [11] for a single controller instance, as standard control traffic recovery time in OpenFlow falls in the order of seconds. As restoration of control traffic delays the restoration of data traffic as well (flow modification might be required at the disconnected switches), *rapid recovery of control paths has utmost importance*. Hence, an OpenFlow framework implementing in-band control with priority queuing and failure recovery functions was proposed [12]. Experiments were conducted, and shown that restoration of either data plane or control plane traffic does not allow achieving the 50 ms recovery time; hence, pro-active approaches are required [13]. Furthermore, as protection does not require controller intervention, it is less dependent on the network topology.

The design of an in-band routing tree spanning all switches and rooted at the single controller was investigated in [14]. Built on previous local route repair mechanisms from IP Fast-ReRoute (e.g., protection routing [15]) a primary tree and secondary (protection) next hops for each switch are designed, where the objective is to minimize the number of unprotected switches upon a single switch or link failure. However, the existence of a secondary next-hop which by-passes the parent switch along the tree is not guaranteed, i.e., full single failure coverage can not be provided. The concept was later extended to in-band trees rooted at different controller instances [16].

B. Resilient Controller Placement

In [17] resilient control plane design was investigated based on partitioning the network and assigning a controller to the centroid of each cluster in order to minimize control latency and the number of controller-less nodes upon link and node failures. It is shown that finding an operating controller upon failure and reassigning the switch might lead to long outages and performance degradation owing to the increased load (i.e., queuing delay) at some controllers. Thus, *pro-actively designing a list of backup controllers* to each switch and auxiliary connections towards them might be beneficial [18].

Resilient controller placement was thoroughly investigated in [19], where the authors considered controller failures, network disruptions affecting in-band communication channels, load-balancing of the traffic on each controller instance and inter-controller latency as objectives. The multi-objective framework provides Pareto-optimal solutions by an exhaustive evaluation of the design space for the combination of these design goals in small- and moderate-size network topologies, and allows network operators to choose the most appropriate solution to their requirements. For large scale topologies where the exhaustive search is not possible owing to memory constraints, or for dynamically changing environments where prompt reconfiguration of controller placement might be needed a heuristic approach is proposed [20] based on Pareto simulated annealing for providing a solution in a timely manner for the placement of a fixed number k of controllers.

Joint optimization of placement of multiple controllers and disjoint in-band control path design to the same controller instance or to two different controller locations was tackled in [5] (similarly as choosing K server locations connected with disjoint path-pairs in optical grids/clouds [21]). Although from a scalability perspective it requires much more forwarding table entries than a single control tree, with this approach all single link and node (switch or controller) failures can be protected.

C. Hypervisor Placement Problem

Although data plane virtualization has been thoroughly investigated, [22] was the first work which introduced flexible control plane virtualization techniques. A comprehensive survey of different network hypervisors was presented in [4], and they were categorized based on their architecture (i.e., centralized or distributed) and their execute platform (i.e., software programs on general compute platforms and special-purpose network elements). As the control plane performance impacts the data plane performance, besides abstraction of resources the hypervisor should provide the isolation of both data- and control plane traffic of different tenants.

In [23], [24] the Hypervisor Placement Problem (HPP) was considered, where all virtual networks – set of virtual switches and their controllers – are known in advance and are given as input (i.e., off-line problem), and the task is to find the number and locations of hypervisors which minimizes maximum or average latency both for all control paths and per individual virtual network. In order to avoid control conflicts and state inconsistencies, *each physical SDN switch should be controlled by a single hypervisor instance* in the distributed single-controller switch architecture [24] investigated in our paper. In [25] it was demonstrated that with fixing hypervisors first and determining controller locations in a second step could lead to a better performance than vice versa [24] in

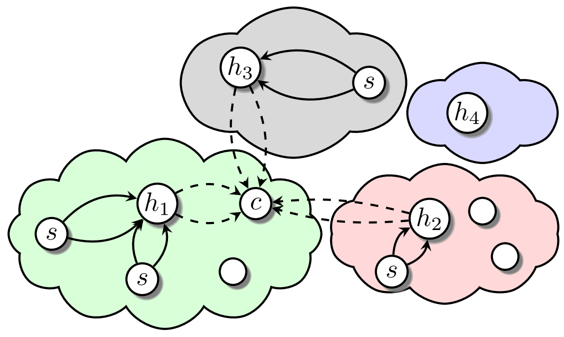


Fig. 1. Network locations (V) hosting both a physical switch and a server to run hypervisors and/or controllers. Four hypervisors $h_1 - h_4$ are deployed, each responsible for a disjoint set of physical switches (marked with clouds). A possible virtual SDN embedding is shown with four virtual switches s and virtual controller location c with the corresponding switch-to-hypervisor (solid) and hypervisor-to-controller (dashed) pre-allocated control paths.

some networks, while a joint optimization of both controller and hypervisor locations would yield to the best performance among the three approaches. Therefore, we will use these observations in our resilient placement problem for the online case where virtual network requests arriving one after the other. The disjoint set of physical switches controlled by their assigned hypervisor are marked with clouds in Fig. 1. After the hypervisors are placed, we find the controller location for a set of switches of a single virtual network request in a second step, and design resilient control channels in the form of link-disjoint paths between them.

Although resilient controller placement is a well investigated topic, to the best of our knowledge, our work is the first which considers resilient hypervisor placement and in-band control channel design pro-actively before the failure occurs. A re-active approach called Dynamic Hypervisor Placement Problem was proposed in [26], where the embedding of virtual networks might be changed owing to a disaster alert [27], which requires the redesign of the control plane and hypervisor assignment. Two approaches were investigated [27], namely when the virtual network to hypervisor assignment is fixed but the hypervisor can be migrated, and when the hypervisor assignments are fixed but the virtual networks can be reassigned to different hypervisors. However, these methods are not suitable for the instantaneous reaction to link failures on the control path.

Minimizing the latency of the switches to the controller(s) or hypervisor(s) as a single objective optimization boils down to the traditional mathematical problem of *facility location*. The closest to our work is [28], where the idea of using set cover for the disjoint-path facility location problem was used for Internet traffic monitoring and content distribution. In [28] each customer must be served by two locations and be connected on shortest disjoint paths. It is proved that no polynomial-time algorithm can guarantee good solutions for the problem, and efficient heuristic algorithms are proposed.

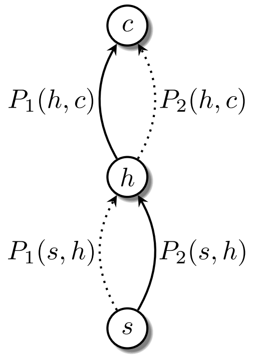
In contrast to [23]–[27] minimizing for (average) control path lengths with complex Integer Linear Programs (ILPs), in our novel resilient self-driving control-plane design we will propose a polynomial-time placement algorithm without using any task-specific objectives, without relying on a predefined number of hypervisor instances or having a priori knowledge about virtual network demands. As a result the resilient control path lengths will not be optimal in our hypervisor placement but still satisfy the SLAs for all (future) virtual networks where possible. Furthermore, in the dynamically changing network environment considered in this paper (i.e., link failures, unknown future vSDN requests) intelligent algorithms and heuristics which leave enough options open and have low computational complexity are required for the flexible reconfiguration of the network when a new placement is needed suddenly, e.g., in reaction to hypervisor failures, disaster alerts or load imbalance on the hypervisors.

D. Function Chain Routing Algorithms

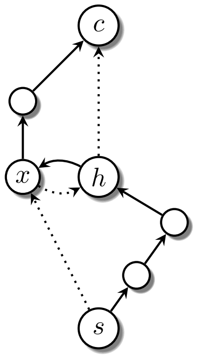
With the proliferation of SDN and virtualization techniques, shortest path routing problems traversing specific types of functions in a given order (called *function chains*) to enforce security policies or visiting ordered sequence of middleboxes [29] become a hot research topic. Although the problem is hard on general network topologies if capacity constraints have to be fulfilled along the links for data plane traffic, in [29] the authors characterize the families of graphs for which polynomial-time algorithms exist. In [30] a Lagrange-relaxation technique was applied for the constrained shortest path problem, i.e., find shortest paths visiting nodes in a specific order. However, the proposed methods cannot guarantee link-disjointness of different segments of the paths; thus, cannot be used for resilient routing.

Resilient allocation of whole functions chains (i.e., joint optimization for placement and control path design) was investigated in [31], [32] against both single link and node failures. In [31] three ILPs were proposed to solve the virtual network function (VNF) placement and resilient control path allocation problem for different failure scenarios. In [32] the service is decomposed into possible realizations of the chain, and in a second step one of the realizations is embedded into the physical substrate using a backtracking algorithm. For the latter problem, backup paths are calculated between two subsequent VNFs, between their backups and between a primary VNF and the backup of its subsequent one in the function chain. With this approach, failure resilience can be ensured upon physical node failures hosting a given VNF. Although these general methods would be applicable in our case, they use complex ILPs with fine-tuned objective functions to solve the problem. In contrast, in our hypervisor placement problem we use a polynomial-time algorithm built on minimal set cover, which can be used in self-driving networks as well. Furthermore, in our control path design problem we can exploit the special structure of paths for the virtual network slices (i.e., our function chains consist of switch-to-hypervisor-to-controller paths), resulting in faster algorithms.

On one hand, fully link- and node-disjoint paths can guarantee 100% resilience against single failures. On the other hand, in several use cases sharing some common elements of the paths might be beneficial. For example, formal language constrained routing can be used to find shortest paths traversing



(a) Logical view



(b) Physical view

Fig. 2. Pre-allocated link-disjoint control paths between the switch and hypervisor, and the hypervisor and the virtual controller in our resilient control plane. The shortest control path in the no failure case is shown with dotted lines. Note that, a single link failure (x, h) might affect multiple paths.

given links (even multiple times) [33], or common links in the disjoint path-pair can be allowed to reduce routing cost while a certain level of availability is maintained [34]. These concepts were extended for link-disjoint problems, where the paths can share minimum, maximum or exactly k nodes, and it was shown that only the upper-bound problem is solvable in polynomial-time [35]. The authors introduced a dynamic program for the above problem variants, as well as for the task when the primary path of the connection is already part of the input, and the goal is to find a secondary path sharing a given number of common nodes with it [35].

E. Our Contribution

In this paper we propose an intelligent self-driving framework which pro-actively designs the resilient control paths for dynamically arriving vSDN requests. First, in Section IV we extend the concepts of resilient (controller) placement and control path design [5] with set cover [28] to our resilient hypervisor placement problem. However, in contrast with [5] we do not fix the number of hypervisors in advance and in contrast with [28] we calculate resilient paths to a single location. In our model the hypervisor locations are selected based on the length of the link-disjoint path-pair to their controlled physical switches, which differentiates our work from [23], [28]. Furthermore, considering dynamic vSDN requests sets apart our placement model from [24], [25], where all requests are given as the input of the problem. Second, after the hypervisor location(s) are fixed, we use a dynamic program extending the concept of [35] to support the control path design of dynamically arriving virtual network requests in Section V.

III. Problem Formulation

As we have seen, pro-active design of backup SDN control paths [11]–[13], [17], [18] are required if we want to meet the strict timing requirements declared in the service level agreements. This statement holds for vSDNs as well, where all control traffic between the virtual switches and its controller needs to traverse (one of) the hypervisor(s). Hence, in this paper we pro-actively design the resilient control paths for each slice.

TABLE I
Notation List for the Resilient Control Plane Design Problem

Notations	Description
$G(V, E)$	directed graph with node set V, link set E, and link lengths $l(e) \in \mathbb{R}^+$
$P(v_1, v_p)$	path (ordered sequence of links) from node v_1 to v_p, with length $l(P) = \sum_{e \in P(v_1,v_p)} l(e)$
`nodeDistance()`	resilient control distance of nodes calculated as the average length of a disjoint path-pair
L	maximum allowed switch-to-hypervisor distance
$h_s \in \mathcal{H}$	hypervisor h_s responsible for physical switch s from the set of hypervisor locations $\mathcal{H}$
$\mathcal{N}(h)$	set of physical switches within control distance $\leq L$ to hypervisor location h
$\mathcal{S}(s)$	set of candidate hypervisor locations which can serve physical switch s (i.e., distance $\leq L$)
$I(h)$	number of switches s within distance $\leq L$ to h without candidate hypervisors (i.e., $\mathcal{S}(s) = \emptyset$)
$\mathcal{R} \subseteq V$	set of nodes representing the virtual switch locations of an embedded virtual network request
D	sum of all switch-to-hypervisor-to-controller distances for virtual network request $\mathcal{R}$
$V_f \subseteq V$	locations hosting network function type f
$d[i][v][f]$	path length from s to v with at most i links that visits network functions in $V_1, \dots, V_f$ in order
$\pi[i][v][f]$	last node on path s to v with at most i links that visits network functions in $V_1, \dots, V_f$ in order

Most of the works assume that control paths are either out-of-band or control traffic has high priority even with in-band control [5], [19], [23], [24], [28]. Built on this fact, the most important difference of our control plane design approach compared to data plane design methods and complexity results there [29] is that we do *not consider link capacities* in our formulations [5], [24]. Note that, without these constraints the control paths of different switches do not influence each other, hence, can be calculated independently, which provides us the opportunity for an intelligent algorithm design and improved complexity results.

A. Resilient Control Plane Model

A *network* is represented by a directed graph $G(V, E)$, where V is the set of nodes and E is the set of links. Each node $v \in V$ represents a physical location in the network with a physical SDN switch. Furthermore, we assume that each location can host virtual network functions such as hypervisors or controllers. Each link $e = (u, v) \in E$ is assigned with a positive length $l(u, v) \in \mathbb{R}^+$ which can represent physical length or delay. A path $P(v_1, v_p)$ is an ordered sequence of links $< e_1 = (v_1, v_2), e_2 = (v_2, v_3), \dots, e_{p-1} = (v_{p-1}, v_p) >$ which provides a communications channel between v_1 and v_p. The total length of a path $l(P)$ is the sum of the link's lengths in the path. We assume symmetric links, i.e., $\forall e = (u, v) \in E : l(u, v) = l(v, u)$, thus, the minimum length (i.e., *shortest*) paths are the same in both directions. Hence, we will calculate the control path only in one direction, but we assume these channels are bidirectional, as well as a link failure affects both directions at the same time. The notations used throughout the paper are summarized in Table I.

In Fig. 2 we present the considered architecture [24] in this paper. We assume pre-allocated link-disjoint control paths between the locations, which can be calculated in advance (consequence of no link capacity constraints), shown in Fig. 2(a). Assuming that $l(P_1(s,h)) \leq l(P_2(s,h))$ and $l(P_2(h,c)) \leq l(P_1(h,c))$, the actual control path in the no failure case is shown with dotted lines. However, if a link failure occurs along this path, the control flow must be rerouted to the other one. Hence, as both lengths impact the overall performance of the virtual network, we will consider the average length of these paths in our algorithms.

Definition 1. *Function* `nodeDistance`(s,h) *represents the resilient* **control distance** *between nodes s and h, defined as*

$$\texttt{nodeDistance}(s,h) = [l(P_1(s,h)) + l(P_2(s,h))]/2.$$

Note that, for the shortest $P^*(s,h)$ path: $l(P^*) \leq$ `nodeDistance`(s,h), where equality holds when both paths in the disjoint path-pair are shortest $P(s,h)$ paths.

B. Reaction to Single Link and Hypervisor Failures

One can observe that in our resilient design both the switch-to-hypervisor and hypervisor-to-controller control channel is a disjoint path-pair by the definition. Although a single link failure can disrupt one of the paths of the path-pairs for both channels if they share a common link (e.g., link (x,h) Fig. 2(b)), a pre-configured control path will still remain to satisfy our resiliency requirement. We assume the same fail-over mechanisms to the pre-calculated backup path upon link failure as previous control plane design approaches for SDN networks [5], [12] or traditional protection approaches. Similarly to data plane traffic, the pre-installed backup resources can be used in the no failure case to forward low-priority traffic or for load-balancing purposes.

As each hypervisor is responsible for a disjoint partition of physical switches [24] as shown in Fig. 1, its failure will affect only the isolation and abstraction of the virtual networks running on these switches. Although it would be possible to rerun a HPP algorithm to recalculate a whole switch-to-hypervisor assignment, creating new partitions of the network controlled by new hypervisor instances at new locations would cause unnecessary disruptions for all tenants with running virtual networks. Instead of this, we either suggest to migrate the hypervisor to a different location within the partition if the disruption can be forecasted [27], or run an HPP algorithm only to the subset of physical switches which were assigned to the failed hypervisor. Note that, owing to the increased shortest path lengths caused by the node failure, there might be multiple new partitions as well.

C. Intelligent Self-Driving Network Operation

In our model self-driving networks are measure, analyze and control themselves and are able to react to changes in the environment (e.g., network failures or new vSDN requests in our case) based on their observations. In such dynamic environment having multiple options to choose from in response to unknown challenges is inevitable, as well as the ability to instantaneously realize this potential. Being prepared for the future – i.e., maximizing the number of options – is a desired property and well-investigated area in communication networks as well (e.g., in resilience), and several definitions agree that such behaviour is considered to be intelligent [1], [3]. On the one hand, this can be achieved in a self-driving manner with algorithms that aim to maximize preparedness as an intrinsic motivation, without relying on comprehensive mathematical models or task-specific objective functions. For example, minimum interference routing [36] keeps bottleneck resources open, which makes more unknown connection requests acceptable in the future without having it as an explicit objective.

On the other hand, without the knowledge of such intrinsic motivation (e.g., in case of our placement problem in Section IV), a weaker approach can be used and formulate the problem in a way that several states satisfy the constraints the self-driving network can choose from in response to changes (in contrast with the single solution of an optimization problem with fine-tuned objective). We note that the algorithm which solves this intelligent formulation can be an optimization method if the parameters should not be tweaked on a case-by-case basis, and if the increased complexity still suits the reaction time requirements of the self-driving network.

IV. Hypervisor Placement with Greedy Set Cover

In this section we propose a polynomial-time algorithm to the resilient hypervisor placement problem, and show that it approximates the optimal solution in terms of the number of hypervisors. Formally, the problem is defined as follows:

Problem 1. Resilient Hypervisor Placement: *Given a network $G(V,E)$, link lengths $l(u,v)$, and constraint L on the maximum switch-to-hypervisor distance. Find a minimum number of hypervisor locations $\mathcal{H} \subseteq V$, where $\forall s \in V : \exists h \in \mathcal{H} \mid$* `nodeDistance`$(s,h) \leq L$.

Our resilient hypervisor placement algorithm is presented in Alg. 1. We assume that the network topology and the maximum switch-to-hypervisor distance[1] is given as the input, and the algorithm returns the hypervisor locations and switch assignment satisfying this constraint. The algorithm consists of three stages as follows. First, in Step (2) to Step (6) the `nodeDistance()` between every node-pair in the topology is calculated and switches within L are given to set $\mathcal{N}(h)$ for every location. Note that, distance could represent several different metrics. In Alg. 1 we use Suurballe's algorithm [37] to get a disjoint path-pair between s and h with minimal average length of the path-pair and use this as `nodeDistance`(s,h) in our resilient placement. However, it can be the distance using a disjoint path-pair which was obtained by minimizing for the shorter path's length (i.e., minimize control plane latency when all links are operational), or by minimizing the longer path's length (i.e., minimum latency upon link failures),

[1]The SLAs declare only the maximum switch-to-hypervisor-to-controller distances. Without any further information about future vSDN requests we set the switch-to-hypervisor distance L as half of the total control distance, which might exclude some acceptable solutions.

Algorithm 1: Hypervisor Placement with Set Cover

```
Input: $G(V, E)$ - network, $l(u, v)$ - link lengths, $L$ - maximum distance;
Output: $\mathcal{H}$ - hypervisor locations, $\forall s \in V : h_s$ - hypervisor assignment;
1  Initialize $\mathcal{H} := \emptyset; \forall v \in V : \mathcal{N}(v) := \emptyset, \mathcal{S}(v) := \emptyset$, where
   $\mathcal{S}(v)$ - hypervisors in range, $\mathcal{N}(v)$ - switches covered;
2  for $h \in V$ do
3      for $s \in V$ do
4          Calculate nodeDistance($s, h$) with Suurballe's algorithm;
5          if nodeDistance($s, h$) $\leq L$ then
6              Add $s$ to set $\mathcal{N}(h)$;
7  while $\exists s \in V : |\mathcal{S}(s)| < 1$ do
8      for $h \in V$ do
9          Calculate importance
           $I(h) := |\{w \in \mathcal{N}(h) : |\mathcal{S}(w)| = \emptyset\}|$;
10     Find best location $h^* := \arg\max_h I(h)$;
11     Add $h^*$ to hypervisors $\mathcal{H} := \mathcal{H} \cup h^*$;
12     for $v \in \mathcal{N}(h^*)$ do
13         Add $h^*$ to switches in range $\mathcal{S}(v) := \mathcal{S}(v) \cup h^*$;
14 for $s \in V$ do
15     Assign switch $s$ to hypervisor $h_s \in \mathcal{S}(s)$, where
       $h_s := \arg\min_h$ nodeDistance($s, h$);
```

or the length difference of the two disjoint paths. However, all of these problem variants are NP-hard [35], and we will show in Section VI-B that their performance gain compared to Suurballe's algorithm is negligible for the price of the increased computational complexity.

In the second stage from Step (7) to Step (13) we perform the greedy set cover, where the base set is the physical nodes V and the set system is the $\mathcal{N}(h)$ at every possible location. In each iteration we greedily select the location which covers the most uncovered switches by the previous iterations, which is calculated in importance $I(h)$. If a location h^* is selected, it serves as a possible hypervisor for all switches $s \in \mathcal{N}(h^*)$, as the `nodeDistance(`s, h^*`)` is at most L; thus, h^* is in the range of s and added to $\mathcal{S}(s)$. Finally, in the last stage in Step (14) to Step (15) we assign the physical switches to one of the hypervisor instances in $\mathcal{H}$, which will perform resource isolation of the virtual switches allocated on them. In Alg. 1 we assign every switch s to the closest hypervisor instance $h_s \in \mathcal{H}$, but a self-driving network could use other objectives as well, e.g., for load balancing or if a constraint is given on the maximum number of switches controlled by a hypervisor.

Lemma 1. *The time-complexity of Algorithm 1 is* $O(|V|^2(|E| + |V| \log_2 |V|))$.

Proof: Calculating `nodeDistance()` with Suurballe's algorithm [37] for all node-pairs and to fill in $\mathcal{N}(h)$ from Step (2) to Step (6) takes $O(|V|^2(|E| + |V| \log_2 |V|))$. For every iteration in Step (7) to Step (13) we have to determine the switches without a hypervisor in range (i.e., $|\mathcal{S}(s)| = 0$), and importance $I(h)$ has to be calculated for each possible location. This process requires to check $\mathcal{N}(h)$ at most in $|V|$ steps for each node, resulting in $O(|V|^2)$. Selecting the best location and updating sets can be done in linear time in $|V|$. Finally, closest hypervisor assignment in Step (14) to Step (15) is $O(|V|)$, which gives an overall complexity of $O(|V|^2(|E| + |V| \log_2 |V|))$ for Alg. 1. ■

In Lemma 1 we have shown that in Alg. 1 we can create a set system $\mathcal{N}(h)$ (with set sizes $1 \leq |\mathcal{N}(h)| \leq |V|$) on the base set V and perform a greedy set cover [28] in polynomial time. Together with the classical results [38], [39] which state that the number of sets in the greedy cover approximates the optimal number of sets within a factor of $\sum_{i=1}^{|V|} 1/i \leq \ln |V| + 1$ for the general unweighted case (by analyzing the structure of the sets lower factor might be achievable), we can make the following observation:

Corollary 1. *Alg. 1 is a polynomial-time (*$\ln |V| + 1$*)-approximation algorithm on the number of hypervisors for Problem 1.*

As a result of Alg. 1, the hypervisor locations in $\mathcal{H}$ are fixed, and together with their assigned switches they partition the network into disjoint parts (shown in Fig. 1), where all control traffic of the partition goes through the corresponding hypervisor.

V. Control Plane Design for Virtual SDNs

In this paper we investigate an on-line problem, i.e., where the virtual SDN network requests are not known in advance and can dynamically arrive and leave the network. Hence, the hypervisor locations were designed for all physical switches in Section IV. In this section we use the already pre-calculated distances in Alg. 1 in our resilient virtual controller placement method in Section V-A for a given set of virtual switches. Once the virtual network is embedded, in Section V-B we present a dynamic program to find minimum length switch-to-hypervisor-to-controller paths both for the unprotected and for the single link failure resilient scenarios.

A. Controller Placement for vSDN Requests

As in our control plane design the capacity limitations of the data plane is not considered, we treat the virtual network embedding algorithm in the data plane as a black box. Therefore, we do not assume anything about the arrival process, holding time, graph structure or the resource requirement of these virtual network requests as they have no influence on our problem. However, the number and location of the virtual switches specifies the hypervisor instance(s) involved in the resilient control plane design and need to be analyzed [23], [24]. Thus, we only assume that the embedding algorithm returns us the virtual switch locations of the new request in the topology, which are given as the input of our virtual controller placement method, and our task is to find a location which minimizes total control distance while bounds the maximum distance for individual switches. Note that, if all virtual switches are controlled by the same hypervisor instance, then the best virtual controller location will be the hypervisor's location for that virtual network.

Formally, for each individual virtual network request we perform the following steps:

- As virtual network embedding is out of the scope of this paper, we assume that the virtual network is embedded on a set of physical switches $\mathcal{R} \subseteq V$, which serves as the input request to our problem.
- For each possible virtual controller location ($\forall v \in V$ in our model) we calculate total control distance D as

$$\sum_{s \in \mathcal{R}} \{\texttt{nodeDistance}(s, h_s) + \texttt{nodeDistance}(h_s, v)\},$$

  where $h_s \in \mathcal{H}$ is the hypervisor for physical switch s.
- Select node $c \in V$ as the virtual controller location if it minimizes D while $\forall s \in \mathcal{R}$: `nodeDistance`(s, h_s) + `nodeDistance`(h_s, c) $\leq 2 \cdot L$. If no such location exists, we pick the one with minimum D and count it as an SLA violation.

Note that, `nodeDistance()` values were already calculated in Alg. 1 and can be reused in the controller placement. After the virtual controller location c of the slice is selected, all node locations are fixed for the virtual network, and all disjoint path-pairs are pre-allocated for the control messages. However, the actual minimum length control path along these pre-allocated channels will depend on the actual link failures in the network, and will be calculated in Section V-B.

B. General Dynamic Program for vSDN Control Paths

In our virtual SDN control path design we are dealing with the problem of minimizing the length of a single control path $P(s, c)$ from a given virtual switch $s \in V$ to the virtual controller $c \in V$ which traverses the hypervisor location $h_s \in \mathcal{H}$ responsible for switch s. Furthermore, with the pre-allocated disjoint path-pairs between $s - h_s$ and $h_s - c$, path $P(s, c)$ can be selected from four different combinations depending on the actual link failure (see Fig. 2). Therefore, instead of dealing only with this particular special case, we propose a more general routing algorithm for the following problem:

Problem 2. Minimum Length Path for Function Chains: *Given a network $G(V, E)$, a source node $s \in V$, a destination node $c \in V$, link lengths $l(u, v)$, and a set of nodes $V_f \subseteq V$ hosting network function type $f = 1, \ldots, k$. Find a minimum-length path $P(s, c)$ between s and c which visits network functions in an ordered sequence from 1 to k.*

Note that, for $k = 1$ and $|V_1| = 1$ without any disjointness requirement the minimum length switch-to-hypervisor-to-controller path can be obtained through two shortest path calculations. However, for the sake of completeness, we will propose an algorithm for the general resilient vSDN control plane design problem, which can be directly used if the hypervisor virtualization functions are decomposed and placed at different locations [22] (i.e., $k > 1$), if the switches have the functionality to support multiple hypervisors in a multi-controller switch architecture [24] (i.e, $|V_1| > 1$), or can be

Algorithm 2: Minimum Length Path for Function Chains

```
Input: $G(V,E)$ - network, $s$ - source, $c$ - target, $l(u,v)$
       - link lengths, $\forall f = \{1,\ldots,k\} : V_f$ - locations of
       function $f$;
Output: $d[][|V|][k]$ - path length with a given sequence
        of network functions, $\pi[][|V|][k]$ - shortest path;
1  $d[0][s][0] = 0$, $d[0][v][0] = \infty$ for $v \in V \setminus \{s\}$;
2  $d[0][v][f] = \infty$ for $f \geq 1$, $\forall v \in V$;
3  for $i = 1$ to $(k+1)(|V|-1)$ do
4      foreach $v \in V$ do
5          for $f = 0$ to $k$ do
6              if $v \in V_f,\ f \geq 1$ then
7                  $d[i][v][f] = \min\Big(d[i-1][v][f],$
8                  $\min_{(y,v)\in E}\big(d[i-1][y][f-1] + l(y,v)\big)\Big)$
9              if $v \notin V_f$ then
10                 $d[i][v][f] = \min\Big(d[i-1][v][f],$
11                 $\min_{(y,v)\in E}\big(d[i-1][y][f] + l(y,v)\big)\Big)$
```

modified in an auxiliary network if there are multiple backup hypervisors which require disjoint control paths [5].

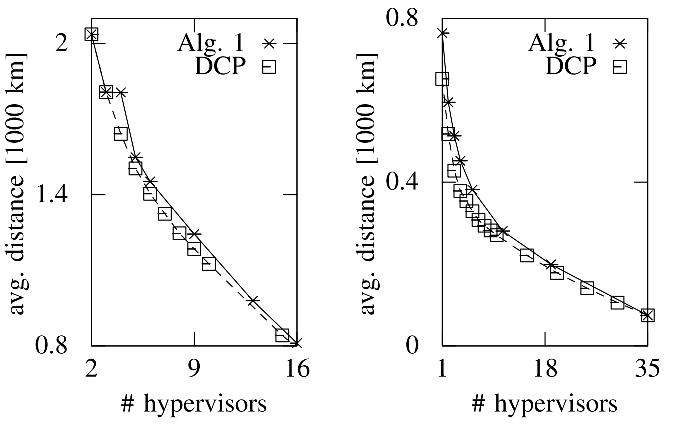
We assume that the $V_f \subseteq V$ location(s) of network function type f is fixed and is given as the input of our routing problem[2]. The presented algorithm follows a similar procedure as the Minimum-Length Link-Disjoint Second Path algorithm presented in [35]. We maintain 3-dimensional arrays d (for the path length) and π (for the paths). For a node $v \in V$ and $f \in [0, k]$, after the ith iteration, we would like the value of $d[i][v][f]$ to describe the length of a minimum-length path between s to v in G that has at most i links and visits network functions from V_1 to V_f in an ordered sequence. Similarly, we maintain in $\pi[i][v][f]$ the previous node in such a path between s to v. The whole procedure is summarized in Algorithm 2.

We first initialize values $d[0][s][0] = 0$, $d[0][v][0] = \infty$ for $v \in V \setminus \{s\}$, $d[0][v][f] = \infty$ for $f \geq 1$ and any node $v \in V$ in Step (1) and Step (2), respectively. In order to calculate the value $d[i][v][f]$ for a node $v \in V$ in iteration i from Step (3) to Step (11), we distinguish between two cases according to whether v is a node hosting network function type f ($v \in V_f$) or not ($v \notin V_f$) as follows.

- $v \in V_f$: We set $d[i][v][f]$ (for $f \geq 1$) as the minimum between its previous value $d[i-1][v][f]$ and $\min_{(y,v)\in E}\big(d[i-1][y][f-1] + l(y,v)\big)$.
- $v \notin V_f$: We set $d[i][v][f]$ (for $f \geq 0$) as the minimum between its previous value $d[i-1][v][f]$ and $\min_{(y,v)\in E}\big(d[i-1][y][f] + l(y,v)\big)$.

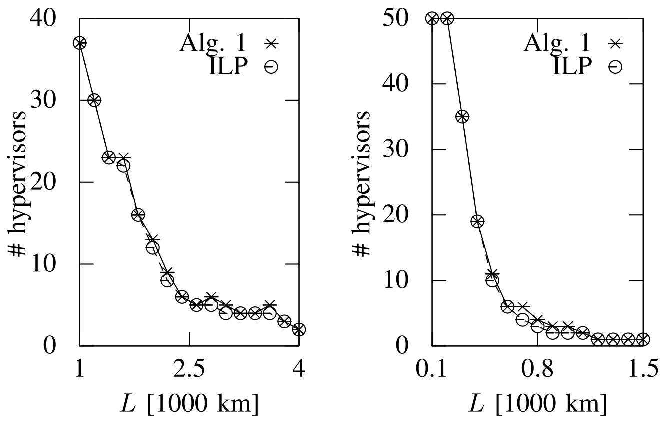
The length of the minimum cost path (if exists) is given by $d[(k+1)(|V|-1)][c][k]$. In order to obtain the corresponding path, we initialize all $\pi[i][v][f]$ to empty, and during the iterations we set it as its previous value or node y in the new

[2]We introduce Alg. 2 on the original $G(V, E)$, but in problems where subsequent network functions can be placed at the same node (as in our case) self-loop links [31] $\forall v \in V$ with $l(v, v) = \epsilon$ length might be added to E.



(a) COST 266 (37 nodes, 57 links) (b) Germany (50 nodes, 88 links)

Fig. 3. Control distances averaged for every switch-to-hypervisor disjoint path-pairs on two real-world topologies [40]. DCP denotes the optimal ILP in [5], while Alg. 1 is our greedy set cover approach.



(a) COST 266 (37 nodes, 57 links) (b) Germany (50 nodes, 88 links)

Fig. 4. Number of hypervisor locations $|\mathcal{H}|$ calculated with the greedy (Alg. 1) and optimal (ILP in Appendix A) set cover approaches with different constraints L on the maximum switch-to-hypervisor control distances.

link (y, v) achieving the minimal value above. The links in the minimum length path can be calculated in a reverse order from the values of π starting from $\pi[(k+1)(|V|-1)][c][k]$.

Lemma 2. *The time-complexity of Algorithm 2 is $O(|V|^2)$ to find switch-to-hypervisor-to-controller paths.*

Proof: Although the path segments between subsequent network functions are simple paths, with adversarial V_f sets all segments might contain $|V|-1$ links, resulting in $(k+1)(|V|-1)$ iterations on the maximum number of links. In each iteration, we consider $k+1$ different traversed network function types for each of the $|V|$ nodes, resulting in a complexity of $O(k^2|V|^2)$ for the general case in Problem 2. In the vSDN control plane design $k=1$, which gives us $O(|V|^2)$, which is slightly worse than two shortest path algorithms $O(|E| + |V|\log_2|V|)$ [37] applicable if $|V_1| = 1$, too. ■

In our resilient design, we will use Algorithm 2 to calculate the shortest switch-to-controller pre-allocated control paths along the disjoint path-pairs both in the no failure case and with single link failures. Furthermore, we will apply Alg. 2 to calculate minimum length switch-to-hypervisor-to-controller control paths in $G(V, E)$ as a benchmark for the unprotected case in Section VI-D.

VI. Experimental Results

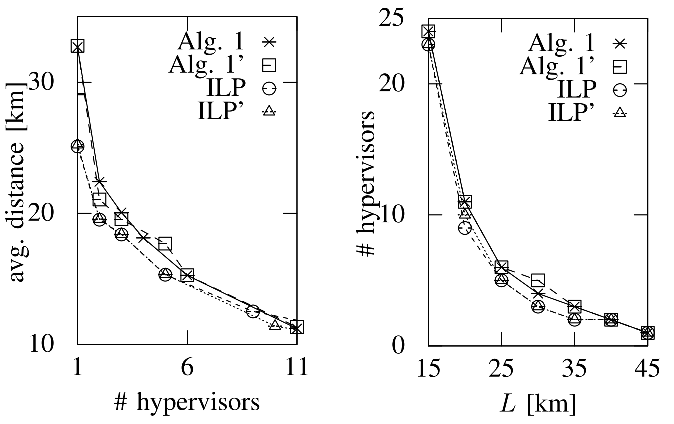
We conducted thorough simulations to demonstrate the efficiency of our polynomial-time algorithms. Section VI-A presents the performance results of Alg. 1 compared to the optimal methods in terms of hypervisor number and average switch-to-hypervisor distance. The effect of different `nodeDistance()` functions on Alg. 1 is investigated in Section VI-B. Section VI-C introduces our virtual network generation and virtual controller placement approach. In Section VI-D we calculate the length of pre-allocated resilient control paths upon single link failures with Alg. 2, and compare it with the unprotected scenario. In these simulations we have selected two larger topologies for comparison (in terms of locations V) from [5] with real-world link lengths [40]. Finally, in Section VI-E we analyzed the average control plane performance of our sub-optimal resilient control plane design compared to an optimal hypervisor placement approach introduced for a static set of virtual network requests in several synthetic and real-world topologies.

A. Resilient Hypervisor Placement with Set Cover

In Fig. 3 we compared Alg. 1 to the optimal solution in terms of control distance `nodeDistance()` in an optimal placement with k locations between every physical switch and the corresponding hypervisor. Because it boils down to the same underlying mathematical problem, for comparison we have implemented the ILP which minimizes average SDN control distance for the Disjoint Control Path (DCP) approach[3] for controller placement with at most k controllers [5]. Note that in DCP the number of controllers k is the input of the problem, while in Algorithm 1 the number of hypervisors is the output of the set cover. Hence, we can not compare the approaches in Fig. 3 for the same controller/hypervisor number all the time. However, we observed that even in the European-size topology in Fig. 3(a) the total distance of the disjoint path-pair in Alg. 1 is at most about 60 km longer (with 9 locations) than the optimal pair (max. 5% increase). We also measured about 60 km distance increase compared to the optimum with Alg. 1 in Fig. 3(b) for 6 locations, but it results in an 18% distance increase for this national topology owing to the shorter physical link lengths between nodes.

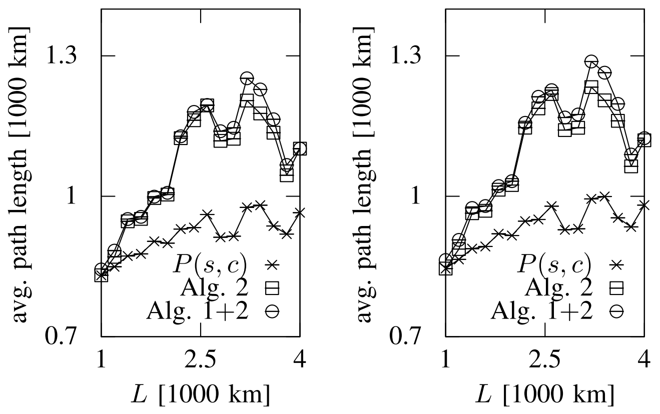
Although DCP provides optimal control distance averaged for all switch-to-hypervisor paths, the maximum distance of some switches was about twice as much as the average for some controller numbers k in our measurements for the real-world networks in Fig. 3. Therefore, it can not guarantee worst case control distances as Alg. 1 can with the pre-defined

[3]Note that, the original formulation in [5] introduces variables to calculate a disjoint path-pair for each pair of nodes. However, as no capacity constraints are considered in the control path design, these paths are independent from each other and can be pre-calculated with `nodeDistance()` as in Alg. 1. Hence, we simplified and speed up their ILP in this way.



(a) Average control distances (b) Number of hypervisors

Fig. 5. Comparison of distance functions in a 100 node 287 link maximum planar graph. Alg. 1 and ILP use Suurballe's algorithm [37] to calculate control distance, while Alg. 1' and ILP' minimize the shorter path's length.



(a) No failures (b) All single link failures

Fig. 6. Control path lengths with and without link failures in the 37 node European network [40] averaged for all virtual control paths in the 1000 virtual network requests (3547 in total). Alg. 1+2 returns the shortest pre-allocated control path with and without link failures in the architecture of Fig. 2.

constraint L. For a fair comparison we formulated a novel ILP which bounds the maximum control path lengths while minimizes the number of required hypervisors, i.e., provides the optimal set cover in Alg. 1. For the sake of completeness, we present the ILP formulation in Appendix A, while our results are shown in Fig. 4. Note that, in the investigated topologies the hypervisor number provided by Algorithm 1 were maximum two instances more than the optimum (but the same or only 1 more in 90% of L values), which is much better than suggested by the approximation ratio in Corollary 1. Hence, we believe that the additional freedom provided by the general formulation of Problem 1 for self-driving networks comes for a manageable price both in terms of hypervisor number and average control distance, while Alg. 1 ensures a low computational complexity as well.

B. Comparison of Different Node Distance Functions

Here we compared the `nodeDistance()` calculation approach where the shorter path's length is minimized from the disjoint path-pair [35] (denoted as Alg. 1' and ILP' in Fig. 5) to Suurballe's algorithm [37] (i.e., average length is minimized) proposed for Alg. 1 and for the ILP in Appendix A. In order to have several options for a disjoint path-pair between the nodes (more than in the 2-connected real-world networks), we have generated a 100 node maximum planar graph [41] with an average nodal degree of 5.74, and set the link lengths uniformly random between 1 and 10 km. As expected, the optimal set cover ILPs performed better in both average control distance in Fig. 5(a) and hypervisor number in Fig. 5(b) than the greedy set cover approaches.

One can observe, that even in this dense communication topology with multiple paths between node-pairs the difference between the two `nodeDistance()` functions is negligible in both metrics. Furthermore, in our implementation the running time of Alg. 1' (paths are calculated with an ILP owing to its computational complexity) was about 250 times longer than Alg. 1 using Suurballe's algorithm. Therefore, we believe that for real-world topologies in disaster areas [42] where network resources should be rapidly evacuated [27] upon receiving a disaster alert (e.g., in milliseconds for earthquakes or in seconds or minutes for tornadoes and hurricanes), or for topologies where placements might be recalculated frequently owing to day-night or hourly traffic fluctuations, Suurballe's algorithm is the better choice to obtain `nodeDistance()` and the pre-allocated control paths in Alg. 1 compared to the ILP formulations with high computational complexity.

C. Embedding Virtual Network Requests

In order to calculate the minimum length virtual switch-to-hypervisor-to-controller control paths for vSDNs in the presence of link failures (shown in Fig. 2) with Alg. 2, in this section we discuss our virtual network request generation and placement approach. In our simulation framework, first we obtained and fixed the hypervisor placement and switch assignment with Alg. 1 for all physical switches (discussed in Section VI-A), without any knowledge of dynamically arriving virtual network requests. In the next step, we generated 1000 virtual network requests, each $\mathcal{R}$ containing between 2 and 5 virtual switches [23], [24], selected uniformly random from V. Finally, we selected the location of the virtual controller independently for each slice as discussed in Section V-A, i.e., minimizing total control distance D, and where possible satisfy constraint $\forall s \in \mathcal{R}$: $\texttt{nodeDistance}(s, h_s) + \texttt{nodeDistance}(h_s, c) \leq 2 \cdot L$. When the constraint cannot be satisfied, there exists at least one switch in $\mathcal{R}$ which violates the maximum switch-to-controller control distance constraint, thus, violates the SLA.

As data-plane embedding algorithms, policies and capacity constraints are out of the scope of the paper, for a fair comparison we assumed that in our simulations all virtual network requests are embedded regardless of whether they satisfy the SLA or not (discussed in Section V-A). Therefore, although we keep track and report SLA violations, their effect is not present in our figures. However, we also implemented the cases when a request should be dropped when the SLA is violated in a real environment, and when the controller placement is

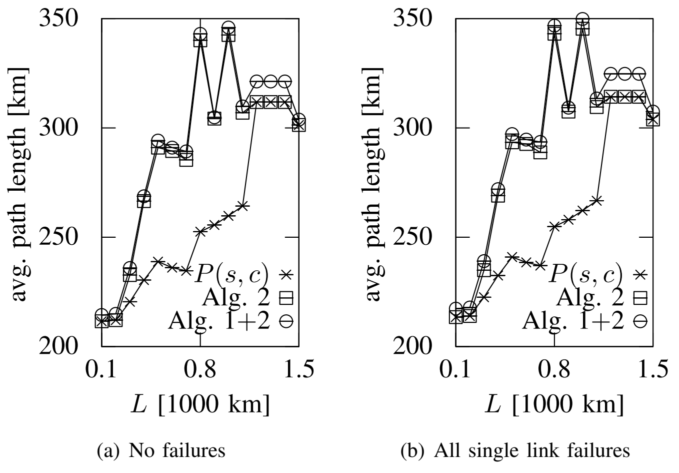


Fig. 7. Control path lengths with and without link failures in the 50 node German network [40] averaged for all virtual control paths in the 1000 virtual network requests (3547 in total). Alg. 1+2 returns the shortest pre-allocated control path with and without link failures in in the architecture of Fig. 2.

done purely on total control distance without maximum control distance constraint. Note that, the differences we observed between these methods were negligible, thus, we followed the procedure proposed in Section V-A which gives us the most comparable results.

D. Minimum Length vSDN Control Paths upon Link Failures

As a result of the vSDN generation in Section VI-C, for each virtual network request we have all locations fixed (i.e., switches, hypervisor, controller), and also we have the pre-allocated disjoint path-pairs which can be used to send control messages for the given slice (shown in Fig. 1). In our simulations we calculated the minimum length control paths depending on the current link failures with three shortest path approaches. First, as a reference on the basic topological properties, we show the minimum length switch-to-controller paths calculated with Dijkstra's algorithm [43], which does not take into account the hypervisor locations ($P(s,c)$ in the figures). As a second approach we use Alg. 2 to find shortest switch-to-hypervisor-to-controller paths upon link failures, which demonstrate the minimum path lengths obtainable with a re-active restoration approach (denoted as Alg. 2 in Fig. 6-7). Finally, we calculate our resilient shortest switch-to-hypervisor-to-controller control paths with Alg. 2 along the pre-allocated disjoint path-pairs provided by Alg. 1 (denoted as Alg. 1+2). When measuring the increased control path lengths upon link failures, we considered all single link failures in the topology in order to avoid distortion of the results because of randomly generated failures. Therefore, our results represent the control path lengths averaged for all switch-to-hypervisor-to-controller paths and for all possible single link failures.

In Fig. 6 we present the measured switch-to-hypervisor-to-controller path lengths averaged to all 3547 control paths in the 1000 virtual network requests for the European-size network both with and without single link failures. One can observe that the average SDN control path $P(s,c)$ length (no hypervisor considered) is about 800–1000 km in this topology without failures in Fig. 6(a). The average length of the paths calculated with Alg. 2 traversing the corresponding hypervisor goes up to 1200 km at $L = 3400$ km with 4 hypervisor instances, while further increases to 1250 km if we use the pre-allocated paths with Alg. 1+2. This 50 km increase in average control path length is what we pay for resilience and instantaneous recovery from link failures compared to Alg. 2. Furthermore, considering all possible single link failures in Fig. 6(b), we observed that the length increase of the control paths is less than 5% on average.

We measured similar trends in Fig. 7 for the 50 node German topology [40]. Remember that as L increases, the number of hypervisor locations decreases (shown in Fig. 4). Therefore, owing to the lower number of hypervisors the average control path length suddenly increases at the points where the greedy set cover returns less locations which covers the whole network within $\leq L$, e.g., at 800 km in Fig. 7. Furthermore, note that even with the same number of hypervisors (e.g., 1 instance for $L = 1200 - 1500$ km), the control path length might decrease as L increases (e.g., at $L = 1500$ km), as the greedy set cover finds a better place for the single hypervisor with a maximum control distance of 1481.2 km. We also note that with a single hypervisor instance the optimal place of the virtual controller will be always at the hypervisor's location in our model [24], which results that $P(s,c)$ and Alg. 2 returns the shortest $P^*(s,h_s)$ path, while Alg. 1+2 provides us `nodeDistance`(s,h_s), clearly showing the length increase we pay in our resilient hypervisor allocation in Alg. 1 compared to the re-active and unprotected case.

Note that, in Fig. 6 for the 37 node European network, the number of SLA violations from 1000 requests decreases from 846 to 100 while the constraint L increases from 1000 km to 3400 km. For the 50 node German network in Fig. 7 all maximum switch-to-controller distances can be satisfied with $L \geq 800$ km (i.e., maximum 1600 km control distance per virtual switch).

E. Average Control Path Lengths

In this section we compare our approach to an optimal hypervisor placement method which finds a given number of k hypervisor locations which minimize the average control path lengths for a pre-defined set of static virtual network requests, without any resilience requirement [24]. Our simulation results are presented in Table II for several SNDLib [40] and Internet Topology Zoo [44] topologies. Note that, in order to obtain real physical distances of the nodes we removed locations without coordinates from the data set, as well as did not consider 1-degree nodes because of the resilience perspective (please, refer to Appendix B for further details). Thus, Table II contains the properties of the resulting topologies.

As the ILP in [24] (and other traditional HPP algorithms [25], too) has completely different objective function and inputs as our self-driving problem definition in the dynamically changing environment, we conducted the following procedure to obtain comparable results of the two approaches:

TABLE II
AVERAGE CONTROL PATH LENGTHS (UPPER PART: SNDLIB [40], MIDDLE PART: INTERNET TOPOLOGY ZOO [44], LOWER PART: SYNTHETIC PLANAR TOPOLOGIES GENERATED WITH LEMON [41]).

	Graph			Average switch-to-hypervisor-to-controller path lengths [km]							
				$L = 1/2$ diameter				$L =$ diameter			
	$\lvert V\rvert$	$\lvert E\rvert$	diam.	k	OPT [24]	Alg. 1+2 (N)	Alg. 1+2 (F)	k	OPT [24]	Alg. 1+2 (N)	Alg. 1+2 (F)
Polska	12	18	812.53	12	256.94	257.29	270.65	3	277.09	331.60	344.43
India	35	80	6417.47	10	2118.21	2510.68	2530.17	4	2194.54	2561.28	2582.02
COST 266	37	57	4036.96	13	856.37	1073.49	1100.17	2	1015.18	1181.21	1204.19
Janos	39	61	5022.24	13	1463.34	1732.99	1763.02	4	1579.45	2009.18	2061.88
Germany	50	88	935.87	12	259.52	312.94	316.58	3	264.84	321.47	326.11
Abilene	11	14	4829.04	11	1562.23	1563.83	1727.01	3	1551.02	1556.05	1800.58
BtEurope	17	30	2716.41	11	719.27	719.40	732.65	4	719.27	719.40	732.65
Quest	19	30	15843.80	13	5554.15	5963.10	6186.57	9	5267.13	5311.14	5598.21
BtNorthAmerica	33	70	4810.12	11	1523.34	1623.28	1640.01	2	1646.28	2109.06	2154.17
Dfn	51	80	778.53	27	254.87	279.16	282.01	2	278.96	314.01	316.97
n100e287 unit	100	287	8	11	3.10	3.76	3.77	4	3.55	4.41	4.42
n100e287 random	100	287	33	23	10.15	12.03	12.09	3	12.29	13.91	13.99
n100e145 unit	100	145	11	68	4.07	4.38	4.43	5	4.78	5.65	5.73
n100e145 random	100	145	58	64	18.53	20.39	20.77	5	22.73	30.10	30.58

- Find a hypervisor placement in $G(V, E)$ with Alg. 1 for a given maximum distance L. As an output it gives us the k number of hypervisors (and their locations).
- Generate 100 virtual network requests $\mathcal{R}_1, \ldots \mathcal{R}_{100}$ with each containing 2 to 10 random virtual switches and find controller locations $c_1, \ldots c_{100}$ based on total control distance D according to Section V-A. We decreased the number of request from 1000 to avoid memory constraints of the ILP [24].
- Calculate the average length of all control paths in all requests along the pre-allocated resilient path-pair with Alg. 2 without link failures and the increased control path lengths upon all single link failures (columns Alg. 1+2 (N) and Alg. 1+2 (F) in Table II, respectively).
- Define the input parameters of the ILP [24] as topology $G(V, E)$, hypervisor number k, and virtual network requests $\{\mathcal{R}_i, c_i\}, i = 1, \ldots 100$. For our use case we have implemented the ILP in [24] with the average latency objective function and without the multi-controller switch constraints. Therefore, the solution of the ILP gives the optimal hypervisor locations for the given set of requests and the optimal average switch-to-hypervisor-to-controller path lengths, denoted as OPT [24] in Table II.

The diameter and the half of the diameter of the topologies were considered as maximum switch-to-hypervisor distances L in Table II in order to produce significantly different number of hypervisors k. However, in this setting if the $L = 1/2$ diameter is considered, the SLA violations are above 50% owing to the shorter allowed control distances. For this tighter constraint with more hypervisor instances and more partitions one would expect decreased control path lengths, which can be observed in most topologies. However, in some Internet Topology Zoo [44] networks one can observe that the more locations result in the same or even increased average control path lengths. First, in case of the BtEurope topology the path lengths are exactly the same for both distance constraints, because there is a central node with high betweenness value which acts as the hypervisor location for most physical switches. Furthermore, our algorithm in Section V-A places the controller to this node for almost every virtual network request. Therefore, increasing the number of hypervisor instances has no additional benefit. Second, more hypervisor instances can cause increased average control path length as well which is counter-intuitive. However, note that owing to the resilience requirement we selected the controller location in each virtual network based on total control distance D from the involved hypervisor instances and not on average control path length. Hence, in topologies like Abilene or Quest where a detour path could be extremely long, this resilient controller placement metric performs poor in terms of average control path length for specific virtual network requests.

We conducted simulations on synthetic 100-node planar topologies generated with LEMON [41] to investigate the effect of the network density and varying link lengths on the performance in Table II, and to analyze the running time on larger topologies. Our observation is that with large number of hypervisors the gap between the optimal average control path length and our resilient average control path length is larger in denser topologies, while with a few hypervisor locations link lengths have higher influence on the performance. Furthermore, the running time for the unit case (i.e., $\forall e \in E : l(e) = 1$) is lower for both algorithms than the uniformly random link lengths, i.e., in the 287-link network it is 8.45 s compared to 9.35 s for Alg. 1+2, and 150.94 s compared to 375.42 s for OPT [24], measured on a machine running Debian Linux version 4.9, with four 2.59 GHz Intel processors and with 8 GB RAM.

The difference between OPT [24] calculated with the knowledge of all virtual network requests and without any resilience requirement, and our resilient Alg. 1+2 without any a priori knowledge of the requests varies significantly in different topologies. In smaller sparse networks where the number of options is limited the difference is negligible and our approach provides instantaneous single link failure recovery for the optimal placement. However, as the size and density of the network increase, the price we pay for resilience and for supporting dynamic behaviour can be up to 20-25% increased average control path length depending on the maximum allowed switch-to-hypervisor distance L.

As in our self-driving framework the network continuously measures and analyzes itself, it can compare the current placement with other possible available solutions satisfying the constraints owing to the general formulation of Problem 1. In case the trade-off between the increased control path length and the benefits of Alg. 1+2 is not acceptable anymore for the currently embedded vSDNs (e.g., the performance gap becomes larger than a threshold), the network can invoke a new resilient control plane design (which might induce some extra migration or resource cost). However, we were focusing on the resilience aspect in this paper and leave such performance-based reactions for future work.

VII. Conclusions

In this paper we introduced a polynomial-time resilient hypervisor placement algorithm based on greedy set cover to satisfy a maximum distance between the physical switches and the hypervisor instances. Our simulation results show that this general method approaches the optimal solution both in terms of hypervisor number and average distance. Furthermore, with defining sets and using a greedy cover instead of defining finely tuned objective functions on a case-by-case basis we make our algorithm applicable in self-driving networks. Although it does not fully satisfy the original definition, we still claim that our approach shows some sort of intelligent behaviour by keeping options (e.g., number of hypervisors) open and let the algorithm decide their outcome instead of giving them as input parameters. As a second contribution we have also proposed a general dynamic program to calculate shortest paths between virtual switches and controllers traversing the corresponding hypervisor locations. Although we introduced our algorithms for the control path design of virtual SDN networks, both our placement and routing algorithm can be applied for more general facility location and arbitrary function chain routing problems without capacity constraints as well.

Appendix

A. Integer Linear Program to Minimize Hypervisor Locations

Here we shortly present the ILP for the optimal set cover in Problem 1. We use binary variable y_j to denote if node j was selected as a hypervisor location and $x_{i,j}$ to represent if switch i is controlled by hypervisor j. We want to minimize the number of hypervisor locations as a primary objective, and among these solutions we are interested in the one with minimal average control distance. Hence, we use a large enough constant λ to stress the primary objective as follows:

$$\min \sum_{j\in V} \{\lambda \cdot y_j + \sum_{i\in V} \texttt{nodeDistance}(i,j)\cdot x_{i,j}\}.$$

The following constraints are required:

$$\forall i \in V\colon \sum_{j\in V} x_{i,j} = 1, \quad (1)$$

$$\forall i,j \in V\colon x_{i,j} \le y_j, \quad (2)$$

$$\forall i,j \in V\colon \texttt{nodeDistance}(i,j)\cdot x_{i,j} \le L. \quad (3)$$

Eq. (1) states that each physical SDN node is controlled by one hypervisor to avoid control conflicts and state inconsistencies [23]. Eq. (2) says that if a switch is controlled by a location, a hypervisor must be placed at that location. Finally, we bound the maximum control distance between any switch and its corresponding hypervisor in Eq. (3).

TABLE III
Modified Internet Topology Zoo [44] Graphs

Graph	$\lvert V\rvert$	$\lvert E\rvert$	no coord.	1-deg.	δ_{orig}	δ_{mod}
Abilene	11	14	0	0	2.54	2.54
BtEurope	24	37	2	5	3.08	3.52
Quest	20	31	0	1	3.10	3.16
BtNorthAmerica	36	76	3	0	4.22	4.24
Dfn	58	87	7	0	3.00	3.13

B. Real-World Topology Modifications

The investigated SNDLib [40] topologies were two-connected with available longitude and latitude coordinates, so no modifications were necessary on those inputs. For the original Internet Topology Zoo [44] graphs (data shown in the first columns of Table III) in a first step we removed nodes with insufficient information (i.e., without coordinates), and in a second step eliminated the 1-degree nodes from the remaining graph. The number of deleted nodes in the subsequent steps is given as no coord. and 1-deg. in Table III. The links having at least one end-node removed were discarded as well, resulting in the topologies used in Table II. Because in both steps low-degree nodes were dropped, the modified topologies used in the simulations have slightly higher average nodal-degree than the original ones, denoted as δ_{mod} and δ_{orig}, respectively, and are shown in the last two columns of Table III.

References

[1] P. Kalmbach, J. Zerwas, P. Babarczi, A. Blenk, W. Kellerer, and S. Schmid, "Empowering self-driving networks," in *ACM SIGCOMM Workshop on Self-Driving Networks (SelfDN)*, Aug 2018, pp. 8–14.

[2] W. Kellerer, A. Basta, P. Babarczi, A. Blenk, M. He, M. Klügel, and A. M. Alba, "How to measure network flexibility? A proposal for evaluating softwarized networks," *IEEE Communications Magazine*, vol. 56, no. 10, pp. 186–192, 2018.

[3] A. D. Wissner-Gross and C. E. Freer, "Causal entropic forces," *Phys. Rev. Lett.*, vol. 110, p. 168702, Apr 2013.

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

[5] P. Vizarreta, C. M. Machuca, and W. Kellerer, "Controller placement strategies for a resilient SDN control plane," in *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*, Sept 2016, pp. 253–259.

[6] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Openflow: Meeting carrier-grade recovery requirements," *Computer Communications*, vol. 36, no. 6, pp. 656 – 665, 2013.

[7] N. L. M. v. Adrichem, B. J. v. Asten, and F. A. Kuipers, "Fast recovery in software-defined networks," in *European Workshop on Software-Defined Networks (EWSDN)*, 2014, pp. 61–66.

[8] P. Thorat, R. Challa, S. M. Raza, D. S. Kim, and H. Choo, "Proactive failure recovery scheme for data traffic in software defined networks," in *IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 219–225.

[9] P. Babarczi, G. Biczók, H. Overby, J. Tapolcai, and P. Soproni, "Realization strategies of dedicated path protection: A bandwidth cost perspective," *Elsevier Computer Networks*, vol. 57, no. 9, pp. 1974 – 1990, 2013.

[10] P. Babarczi, A. Pašić, J. Tapolcai, F. Németh, and B. Ladóczki, “Instantaneous recovery of unicast connections in transport networks: Routing versus coding,” *Elsevier Computer Networks*, vol. 82, pp. 68 – 80, 2015, Special Issue on Robust and Fault-Tolerant Communication Networks.

[11] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, “Fast failure recovery for in-band openflow networks,” in *Proc. IEEE Design of reliable communication networks (DRCN)*, 2013, pp. 52–59.

[12] ——, “In-band control, queuing, and failure recovery functionalities for openflow,” *IEEE Network*, vol. 30, no. 1, pp. 106–112, 2016.

[13] R. Khalili, Z. Despotovic, and A. Hecker, “Flow setup latency in sdn networks,” *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2631–2639, 2018.

[14] N. Beheshti and Y. Zhang, “Fast failover for control traffic in software-defined networks,” in *Global Communications Conference (GLOBECOM), 2012 IEEE*. IEEE, 2012, pp. 2665–2670.

[15] K.-W. Kwong, L. Gao, R. Guérin, and Z.-L. Zhang, “On the feasibility and efficacy of protection routing in ip networks,” *IEEE/ACM Transactions on Networking (ToN)*, vol. 19, no. 5, pp. 1543–1556, 2011.

[16] Y. Jiménez, C. Cervelló-Pastor, and A. J. García, “On the controller placement for designing a distributed sdn control layer,” in *2014 IFIP Networking Conference*, 2014, pp. 1–9.

[17] Y. Zhang, N. Beheshti, and M. Tatipamula, “On resilience of split-architecture networks,” in *2011 IEEE Global Telecommunications Conference (GLOBECOM)*, 2011, pp. 1–6.

[18] B. P. R. Killi and S. V. Rao, “Optimal model for failure foresight capacitated controller placement in software-defined networks,” *IEEE Communications Letters*, vol. 20, no. 6, pp. 1108–1111, 2016.

[19] 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 *Teletraffic Congress (ITC), 2013 25th International*. IEEE, 2013, pp. 1–9.

[20] 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*, vol. 12, no. 1, pp. 4–17, 2015.

[21] C. Develder, J. Buysse, B. Dhoedt, and B. Jaumard, “Joint dimensioning of server and network infrastructure for resilient optical grids/clouds,” *IEEE/ACM Transactions on Networking (TON)*, vol. 22, no. 5, pp. 1591–1606, 2014.

[22] 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)*, 2015, pp. 397–405.

[23] A. Blenk, A. Basta, J. Zerwas, and W. Kellerer, “Pairing sdn with network virtualization: The network hypervisor placement problem,” in *IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, 2015, pp. 198–204.

[24] A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer, “Control plane latency with sdn network hypervisors: The cost of virtualization,” *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 366–380, 2016.

[25] 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*, vol. 15, no. 2, pp. 840–853, 2018.

[26] A. Blenk, “Towards virtualization of software-defined networks: Analysis, modeling, and optimization,” PhD Dissertation, Technische Universität München, 2018.

[27] M. Tornatore, P. Babarczi, O. Ayoub, S. Ferdousi, R. Lourenco, J. Zerwas, A. Blenk, M. Klügel, and W. Kellerer, “Alert-based network reconfiguration and data evacuation,” in *Guide to Disaster-resilient Communication Networks*, J. Rak and D. Hutchinson, Eds. Springer, 2020, ch. 14, pp. 353–377.

[28] D. S. Johnson, L. Breslau, I. Diakonikolas, N. Duffield, Y. Gu, M. Hajiaghayi, H. Karloff, M. G. C. Resende, and S. Sen, “Near-optimal disjoint-path facility location through set cover by pairs,” *Operations Research*, vol. 68, no. 3, pp. 896–926, 2020.

[29] S. A. Amiri, K. Foerster, R. Jacob, M. Parham, and S. Schmid, “Waypoint routing in special networks,” in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, 2018, pp. 1–9.

[30] A. Van Bemten, J. W. Guck, P. Vizarreta, C. M. Machuca, and W. Kellerer, “Larac-sn and mole in the hole: Enabling routing through service function chains,” in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018, pp. 298–302.

[31] A. Hmaity, M. Savi, F. Musumeci, M. Tornatore, and A. Pattavina, “Virtual network function placement for resilient service chain provisioning,” in *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*, 2016, pp. 245–252.

[32] M. T. Beck, J. F. Botero, and K. Samelin, “Resilient allocation of service function chains,” in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016, pp. 128–133.

[33] C. Barrett, R. Jacob, and M. Marathe, “Formal-language-constrained path problems,” *SIAM Journal on Computing*, vol. 30, no. 3, pp. 809–837, 2000.

[34] J. Yallouz and A. Orda, “Tunable qos-aware network survivability,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 139–149, 2017.

[35] J. Yallouz, O. Rottenstreich, P. Babarczi, A. Mendelson, and A. Orda, “Minimum-weight link-disjoint node-“somewhat disjoint” paths,” *IEEE/ACM Transactions on Networking*, vol. 26, no. 3, pp. 1110–1122, June 2018.

[36] K. Kar, M. Kodialam, and T. V. Lakshman, “Minimum interference routing of bandwidth guaranteed tunnels with MPLS traffic engineering applications,” *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 12, pp. 2566–2579, Dec 2000.

[37] J. W. Suurballe and R. E. Tarjan, “A quick method for finding shortest pairs of disjoint paths,” *Networks*, vol. 14, no. 2, pp. 325–336, 1984.

[38] L. Lovász, “On the ratio of optimal integral and fractional covers,” *Discrete Mathematics*, vol. 13, no. 4, pp. 383–390, 1975.

[39] D. S. Johnson, “Approximation algorithms for combinatorial problems,” *J. Comput. Syst. Sci.*, vol. 9, no. 3, pp. 256–278, Dec. 1974.

[40] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, “SNDlib 1.0–Survivable Network Design Library,” in *Proc. INOC*, 2007.

[41] “LEMON: A C++ library for efficient modeling and optimization in networks.” [Online]. Available: http://lemon.cs.elte.hu

[42] A. Pašić, R. Girão-Silva, F. Mogyorósi, B. Vass, T. Gomes, P. Babarczi, P. Revisnyei, J. Tapolcai, and J. Rak, “eFRADIR: An Enhanced FRAmework for DIsaster Resilience,” *IEEE Access*, vol. 9, pp. 13 125–13 148, 2021.

[43] T. Gomes, L. Jorge, R. Girão-Silva, J.Yallouz, P. Babarczi, and J. Rak, “Fundamental schemes to determine disjoint paths for multiple failure scenarios,” in *Guide to Disaster-resilient Communication Networks*, J. Rak and D. Hutchinson, Eds. Springer, 2020, ch. 17, pp. 429–453.

[44] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, “The internet topology zoo,” *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, October 2011.

Péter Babarczi (M’11) received the M.Sc. and Ph.D. (*summa cum laude*) degrees in computer science from the Budapest University of Technology and Economics (BME), Hungary, in 2008 and 2012, respectively. In 2017-2019 he was an Alexander von Humboldt Post-Doctoral Research Fellow with the Chair of Communication Networks at the Technical University of Munich, Germany. He is currently working as an Assistant Professor with the Department of Telecommunications and Media Informatics at BME. His current research interests include intelligent self-driving networks, multi-path Internet routing, 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 Post-Doctoral Research Fellowship of the Alexander von Humboldt Foundation in 2017. Since 2020, he is the lead researcher of an OTKA FK Young Researchers’ Excellence Programme supported by the National Research, Development and Innovation Fund of Hungary.