Intelligent Control Plane Design for Virtual Software-Defined Networks

Péter Babarczi, Ferenc Mogyorósi, and Alija Pašić
Department of Telecommunications and Media Informatics, Faculty of Electrical Engineering and Informatics,
Budapest University of Technology and Economics, Műegyetem rkp. 3., H-1111 Budapest, Hungary

***Abstract*—In virtual Software-Defined Networks (vSDN) – where multiple tenants share the same physical SDN topology – control plane resilience has utmost importance. Although previous network hypervisor placement methods were able to provide single link and hypervisor failure resilience with short control paths, the placement had to be frequently adjusted if the vSDN requests changed dynamically. In this paper we propose two approaches which follow an intelligent design principle, i.e., besides maximizing acceptance ratio of current vSDN requests they also prepare the placement for the future. Our first method uses a representative request set and the network flexibility metric to maximize preparedness to unknown failures and traffic changes. Our second approach proposes a self-adjusting control plane, where for each new request set a single hypervisor instance and its corresponding control path might be migrated to a new location. We conduct thorough simulations to investigate the performance of the approaches, and demonstrate that they significantly outperform previous methods from the literature.**

***Index Terms*—preparedness, intelligent algorithms, virtual software-defined networks, resilient hypervisor placement**

I. Introduction

Owing to the continuously increasing complexity of communication networks, rapid recovery from failures is only possible through automated mechanisms which doesn't require human interaction. In a self-driving operation the network continuously measures its key parameters, analyzes them and invokes the necessary control actions in response to changes in the environment to drive itself into a "well-prepared" stable state. In order to enable rapid evaluation of complex scenarios and thus, ultra-fast reaction to a wide-variety of environmental changes, the used mechanisms should be as general as possible, operate without using any control plane messages, and most importantly, they should be *intelligent*.

In biology intelligence is often recognized as the ability of an organism to adapt to the environment through observing it, learning from these observations and making a decision to survive. Those plants and animals who are prepared and can react to more challenges will have better chance to live. Several mathematical models were proposed to quantify such intelligence in engineering as well. In [1] it is defined as a physical force which drives the system into a state with maximal entropy to maximize its future freedom of action[1]. Similarly, the information-theoretic tool-set of *empowerment* [2] models the intelligence as the channel capacity between an agent's sensors and actuators, where a state with higher empowerment value – i.e., with more reaction options – is preferred. This metric was successfully applied both in robotics [3] and in communication networks [4].

Based on the above definitions the meaning of intelligence boils down to the following two objectives:

(i) adapt to the current environment as well as possible;
(ii) *being prepared* for the unseen future with maximizing the number of reaction options.

In the former case we need to continuously react to every change which is impossible in a dynamic environment, while the latter requires a huge number of future changes to be considered which might lead to the intractability of the problem.

In this paper we apply these design objectives on the resilient control plane design of virtual Software-Defined Networks (vSDN), where each switch-to-controller path of a vSDN request must traverse a hypervisor instance, which ensures security and separation functions of different tenants. First, we propose an intelligent hypervisor placement algorithm which not only provides an optimal single-link failure resilient allocation for the current set of demands [5], but also allocates a backup hypervisor for each switch against hypervisor failures [6]. As a significant improvement on [6] our novel approach selects a placement with maximum *flexibility*, i.e., with the maximum number of possible reactions to fulfill Objective (ii). Therefore, with a careful design frequent reconfigurations can be avoided. Second, we propose a self-adjusting approach to investigate Objective (i), where after each change a single hypervisor instance can be migrated to a new location, thus, continuously adapting the network to the optimal state without disrupting the current operation. We demonstrate the benefits of both approaches in our simulations.

The rest of the paper is organized as follows. Section II summarizes the related work on metrics to measure preparedness and on self-adjusting algorithms. We formulate the vSDN placement and control plane design problem in Section III. Our intelligent placement algorithm and self-adjusting hypervisor migration strategy are proposed in Section IV and Section V, respectively. Our simulations results are presented in Section VI, while the paper is concluded in Section VII.

II. Background and Related Work

This section is devoted to give a brief overview of different design concepts to maximize preparedness, focusing on their application to vSDNs. Section II-A introduces two

[1]Which definition agrees with the quote attributed to Stephen Hawking: "Intelligence is the ability to adapt to change."

possible metrics which were proposed to measure the freedom of action. Section II-B introduces examples of self-driving operation in networks leveraging self-adjusting data structures.

A. Potential and Realization: Empowerment versus Flexibility

Although a huge number of available options is necessary for an intelligent algorithm design, in most cases it is not sufficient to provide network preparedness. Empowerment measures the potential influence a network has on its environment. Better actuators mean more ways to change the environment by the agent, while with better sensors more action outcomes can be perceived. In communication networks where actions are discrete (e.g., number of flows routed or number of links reconfigured [4]), the empowerment metric simplifies to the (logarithm of) unique reachable states, thus, maximizes available reaction options. However, empowerment does not measure whether this potential can be realized or not within a given cost and time constraint [2], [4], which has utmost importance in order to avoid any interruption in the provided service, e.g., a failure in the optical-layer must be restored before the IP-layer senses the disruption and the routing algorithm starts reconfiguration.

Therefore, the *network flexibility* metric was proposed [7], [8] to measure this responsiveness of the system to a given set of environment changes. Dynamic controller placement was investigated in [9], and the authors showed how often changes in the traffic pattern require controller migration in order to minimize average control path latency. It was argued that – counter-intuitively – using more controller instances not necessarily leads to a more flexible network. In [10] flexibility of different hypervisor migration strategies was investigated against disaster alerts, and different architectures were analyzed depending on the disaster's time scale. However, no pre-allocated backup control paths or hypervisors were calculated. Furthermore, flexibility cost models for network function migration [11] was investigated in [12], but the specifics of resilient vSDNs were not considered in their model either.

B. Self-Adjusting Network Operation

Distributed self-adjusting tree networks were proposed in [13] which can dynamically adapt the topology in response to changes in the traffic pattern by leveraging the potential of state-of-the-art hardware technologies, e.g., free-space optics in data centers or optical circuit switches in wide-area networks. Based on the idea of self-adjusting binary search tree (splay-tree) data structures [14], in communication networks frequent communication pairs are brought closer to each other by reconfiguring edges; thus, the routing cost is significantly reduced. Furthermore, for general (non-tree) topologies an empowerment-based model was proposed in [4], and shown that more requests can be routed if edge reconfigurations are possible. Clearly, the efficiency of these approaches highly depend on the cost of reconfiguration versus the routing cost.

Besides network topology adaptation, self-adjusting data structures – self-adjusting lists [15] to be specific – can be used for packet classification in communication networks [16]. However, in packet classification the extra constraints of rule priorities (i.e., partially ordered list) have to be fulfilled as well. The idea is to move the frequently accessed match-action rules to the top of the routing table (until possible without violating the partial order), thus, packet classification speed can be improved for demand matrices with high traffic locality.

III. Resilient Hypervisor Placement Problem

Here we formulate the problem of resilient *control plane* design (i.e., controller and hypervisor placement) for vSDNs. We assume that control traffic has priority over data traffic, thus, similarly to previous approaches [5], [6] no link capacity constraints are considered in the model. As the proposed algorithms extend the single-link and hypervisor failure resilient *static latency-aware resilient hypervisor placement problem* (SHPP) [6], first we shortly summarize it in Section III-A. In order to improve SHPP, we extend the original model to a multi-objective optimization by minimizing the maximum number of virtual switches served by hypervisors and vSDN requests managed by controllers, respectively; thus, mitigating the after-failure migration burden on the control plane, described in Section III-B. Finally, our hypervisor migration model is discussed in Section III-C.

A. Static Latency-Aware Resilient Hypervisor Placement

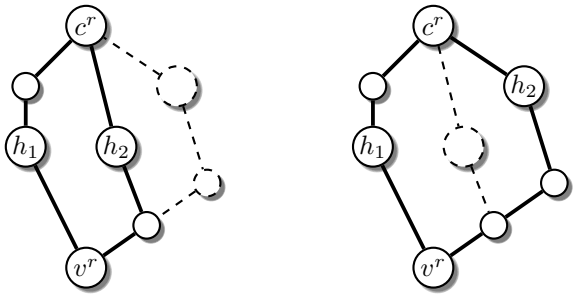
The physical topology is modeled as a graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ with nodes $v \in \mathcal{V}$ connected by undirected edges (links) $e \in \mathcal{E}$ with latency function $l(e)$ based on their lengths. Each node hosts a physical SDN switch ($\mathcal{S} = \mathcal{V}$), while potential hypervisor and controller locations are given in the sets $\mathcal{H} \subseteq \mathcal{V}$ and $\mathcal{C} \subseteq \mathcal{V}$, respectively. In SHPP the current set of vSDN requests $\mathcal{R}$, as well as $\forall r \in \mathcal{R}$ the data plane embedding is given as part of the input as a connected subgraph $\mathcal{V}^r \subseteq \mathcal{V}$, which contains both the physical nodes of the embedded virtual switches as well some intermediate physical nodes the virtual links traverse [6]. Our objective is to maximize the *acceptance ratio* for $\mathcal{R}$ with at most $k = |\mathcal{H}^*|, \mathcal{H}^* \subseteq \mathcal{H}$ active hypervisors:

$$\max\Big\{\sum_{r\in\mathcal{R}} a_r\Big\}, \tag{1}$$

where a_r is one if r is acceptable and zero otherwise. A request r can be accepted if $\forall v^r \in \mathcal{V}^r$ there exist:

- a unique controller location $c^r \in \mathcal{C}$;
- two active hypervisors $h_1, h_2 \in \mathcal{H}^*$ which serve r,
- using two link-disjoint paths $p_1(v^r, h_1, c^r)$ and $p_2(v^r, h_2, c^r)$ from v^r to c^r traversing the hypervisors (e.g., in Figure 1), where the total latency of both paths is lower than the *maximum global control path latency constraint* L: $\forall i \in \{1,2\} : \sum_{e\in p_i(v^r,h_i,c^r)} l(e) \leq L$.

For an efficient implementation placements which satisfy all above constraints are pre-calculated and stored in the set of quartets $\mathcal{Q} = \{(c, h_1, h_2, v^r)\}$. We refer to the Integer Linear Program (ILP) formulation in [6] which provides the optimal solution for the above SHPP problem as ILP$_a$.



(a) Before migration (b) After migration

Fig. 1. Link-disjoint control paths $p_1(v^r, h_1, c^r)$ and $p_2(v^r, h_2, c^r)$ (bold lines) between a switch and the vSDN controller traversing two active hypervisors. Backup can be migrated without disrupting the service.

B. Minimize Maximum Hypervisor and Controller Load

For better load balancing the number of vSDN requests per hypervisor should be limited. Previous resilient hypervisor placement approaches [5], [6], [17] introduced high imbalance between hypervisor nodes, having a central hypervisor location used by most requests. Introducing upper bound on controller load [18], [19] and delay [20] was already investigated for SDNs, but hypervisors were not considered. Therefore, besides minimizing the number of requests served by a controller, we extend our model with minimizing the maximum hypervisor load as well.

Our intelligent algorithm design goal in Objective (ii) is to *keep bottleneck resources open as long as possible* [21], and assign vSDN requests to hypervisor and controller locations with lower load from the ones which satisfy the maximum global control path latency constraint[2]. As a more balanced load distribution among hypervisors and controllers can significantly improve the reliability of vSDNs (and also increases the placement's flexibility), we integrate multiple objective functions into ILP$_a$. Our secondary objective function besides Eq. (1) minimizes the maximum load of any hypervisor:

$$\min\Big\{\max_{h\in\mathcal{H}} load(h)\Big\}, \tag{2}$$

where $load(h)$ is the *number of switches* managed by h. The third objective minimizes the maximum load of any controller:

$$\min\Big\{\max_{c\in\mathcal{C}} load(c)\Big\}, \tag{3}$$

where $load(c)$ is the *number of vSDN requests* managed by c. These objective functions may be in conflict with each other, i.e., improving one may result in a deterioration of another. However, maximizing acceptance ratio is always our primary objective, and any additional objectives are considered subsequently under the constraint that the acceptance ratio remains unchanged for $\mathcal{R}$. This ensures that the best hypervisor placement $\mathcal{H}^*$ has maximal acceptance ratio with the most balanced distribution of vSDN control load.

[2]Remember that the candidate locations (c, h_1, h_2, v^r) satisfying L are already stored in $\mathcal{Q}$, hence, no additional preparation required.

C. Hypervisor Migration

In this paper we follow previous control plane migration models [5], [6] to deploy the new placement, in which only hypervisors (and their respective control paths) are relocated, while controllers and the switches remain at the same node as shown in Figure 1. The vSDNs with failed nodes or disrupted data planes can be submitted as a future request after the control plane was recovered. Furthermore, owing to our resilient hypervisor placement, each switch is served by a primary and backup hypervisor which are continuously synchronized, thus, the two instances are equal. Therefore, if one of them fails or temporarily shut down and migrated to a new location, the other still can serve the vSDN request(s). Upon migration, we distinguish between two strategies:

- **Stateless:** A hypervisor can be invoked at a new node while shut down at the old independently.
- **Stateful:** State have to be transferred from the primary hypervisor location h_1 to the new h_2 backup location[3].

IV. Intelligent Hypervisor Placement Heuristic

We propose a novel intelligent heuristic approach which selects a placement $\mathcal{H}' \subseteq \mathcal{H}$ with minimal number of k hypervisors satisfying the latency constraints to achieve Objective (ii). Our previous research showed [6] that using a *representative request set* $\mathcal{R}_{rep}$ (which contains the most probable but unknown future requests) is already beneficial to obtain a hypervisor placement with high acceptance ratio. By considering additional metrics, the novel heuristic is able to deliver solutions with significantly improved preparedness and consistency compared to its previous counterparts [5], [6].

A. Hypervisor Placement Heuristic

Given the quartets $\mathcal{Q} = \{(c, h_1, h_2, s)\}$ and a representative set $\mathcal{R}_{rep}$, the objective in Algorithm 1 is to find the minimum number of hypervisors and the assignment of hypervisor-pairs to physical SDN switches. In Phase 1 we apply a greedy set cover as in [6], but we extended it with an additional MAXPREP metric to maximize preparedness (discussed in Section IV-B), and in each iteration the hypervisor with the highest overall score is selected in Step 3. In Phase 2 we make post-process and check whether we can drop any hypervisor while still maintaining full coverage, i.e., $\forall s \in \mathcal{S}$ at least one $\{h_1, h_2\}$ pair remains in $\mathcal{Q}$. Finally, in Phase 3 – in contrast with the random selection in [6] – we assign the hypervisor pair to the switch which provides the maximum number of controller locations (according to $\mathcal{Q}$) in order to maximize assignment options for future requests. These two novel parts compared to Algorithm 1 from [6] are denoted with bold.

B. Intelligent Hypervisor Placement and Candidate Selection

Note that, the SHPP problem was solved with a two-step approach in [6]. First, the number of hypervisors required to have at least one quartet in $\mathcal{Q}$ for each physical switch

[3]Owing to our control path design, the distance between h_1 and h_2 is always $\leq L$, both before and after migrating one of them.

Algorithm 1: Hypervisor Placement Heuristic

Input: $\mathcal{Q}$- set of feasible quartets $\{(c, h_1, h_2, s)\}$; $\mathcal{R}_{rep}$- representative request set;
Output: $\mathcal{H}'$ - hypervisor locations; $\forall s \in \mathcal{S} : \mathcal{H}_s = \{h_1, h_2\}$ - switch-to-hypervisor assignment;

Initialize $\mathcal{H}' := \emptyset$;
`// Phase 1: Perform greedy set cover`
while $\exists s \in \mathcal{S}$ *not covered by* $\mathcal{H}'$ **do**
  Find $h^* \in \mathcal{H} \setminus \mathcal{H}'$ for which $\mathcal{H}' \cup h^*$ the score MAXCOVER + **MAXPREP** is maximal;
  Add h^* to hypervisors $\mathcal{H}' := \mathcal{H}' \cup h^*$;
`// Phase 2: Post-processing`
for $h \in \mathcal{H}'$ **do**
  if $\mathcal{H}' \setminus \{h\}$ *is a cover for* $\mathcal{S}$ **then**
    $\mathcal{H}' := \mathcal{H}' \setminus \{h\}$;
`// Phase 3: Switch-to-hypervisor assignment`
For every switch $s \in \mathcal{S} \setminus \mathcal{H}'$ select $\mathcal{H}_s = \{h_1, h_2\} \in \mathcal{H}'$ that provides the **maximum number of possible controller locations** for s;

s was minimized with a Greedy heuristic, which in each iteration selected a hypervisor position h that maximized the number of newly covered switches (denoted as MAXCOVER) based on a set cover by pairs algorithm. In order to improve the performance of the algorithm, it was repeated 400 times to generate *candidate placements* [22], from which one was randomly chosen with minimum hypervisor number k. In a second step, using k from the Greedy heuristic as input the ILP$_a$ was formulated which used a representative request set $\mathcal{R}_{rep}$ in order to select a hypervisor placement which maximized Eq. (1).

In order to satisfy both Objective (i) and (ii) with a *one-step approach*, we introduce a second metric besides MAXCOVER in Algorithm 1, called MAXPREP, which maximizes Eq. (1). The MAXPREP value consists of four components which are normalized and combined into a score. The first two components maximize the number of quartets in $\mathcal{Q}$ (i.e., possible control structures) for the current set of demands $\mathcal{R}$ and a representative set $\mathcal{R}_{rep}$, respectively. The third component represents the realization potential of these placements discussed in Section II-A. We evaluate *flexibility* $\varphi_r(C, T)$ for each request $r \in \mathcal{R}$, which indicates whether r can be served with $\leq C$ migration cost and $\leq T$ migration time or not [9]. Placements with higher flexibility means better preparedness. In case of stateless migration we set $C = \{1, \ldots, k\}$; $T = \infty$, while for stateful migration $C = \{1, \ldots, k\}$; $T = \sum_{e \in p_{(h_1, h_2)}} l(e)$, i.e., we estimate the migration time with the shortest path latency between h_1 and the new location of h_2.

Finally, a Gaussian random variable was added as a fourth component to increase the likelihood of finding candidate placements with close-to-optimal acceptance ratio through

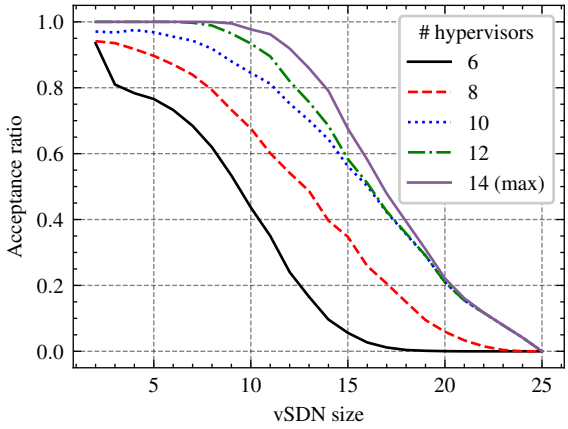


Fig. 2. The effect of active hypervisor instances on the acceptance ratio in the Italy network ($L = 0.5$) calculated with the ILP$_a$ method [6].

repeated (e.g., 400) runs [6], [22] of Algorithm 1 using a novel *intelligent candidate selection* approach. First, the candidate placement list is pruned of duplicates and placements not using the minimal number of k hypervisors. Second, instead of random selection the best solution in terms of acceptance ratio on $\mathcal{R}_{rep}$ is selected . We demonstrate in Section VI-B that with the novel MAXCOVER + MAXPREP score and with this intelligent candidate selection process based on $\mathcal{R}_{rep}$ we can significantly outperform – in fact, *producing optimal solutions* – the Greedy heuristic [6], which relies purely only on MAXCOVER and a random candidate selection.

V. Self-Adjusting Hypervisor Migration Strategy

Although Algorithm 1 was designed to be intelligent through optimizing for the current requests and also maximizes preparedness according to Objective (i) and (ii), respectively, continuously calculating a hypervisor placement from scratch to maximize acceptance ratio in a dynamic environment where demands are arriving and leaving the network frequently is not possible owing to its computational complexity [6]. Therefore, in this section we propose a self-adjusting method, which instead of a fully new solution slightly adapts the hypervisor locations after each change in the request set $\mathcal{R}$ and tries to keep the placement as accurate as possible.

A. Resilient Self-Adjusting Hypervisor Migration

Owing to our resilient hypervisor design it is enough to follow a simple rule in the proposed self-adjusting control plane: when the request set $\mathcal{R}$ is changing (either because of traffic load or single-link failure), we adjust the location of *at most a single hypervisor* at a time. With this operation existing vSDN requests in the network can continuously operate without any service disruption (for the price of reduced reliability during the migration process), while the new hypervisor placement can improve the acceptance ratio compared to a static solution.

In our simulations we relocate the single hypervisor instance which improves the latency values the most, or we leave the current placement as it is if no such instance exists. However,

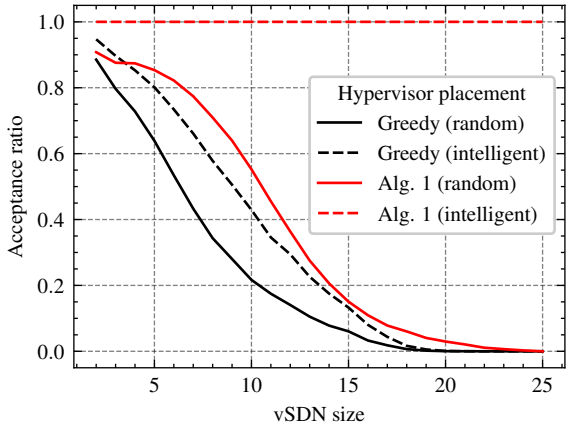


Fig. 3. Comparison of the Greedy heuristic in [6] with Algorithm 1 using random and intelligent candidate selection in the Italy network ($L = 0.6$).

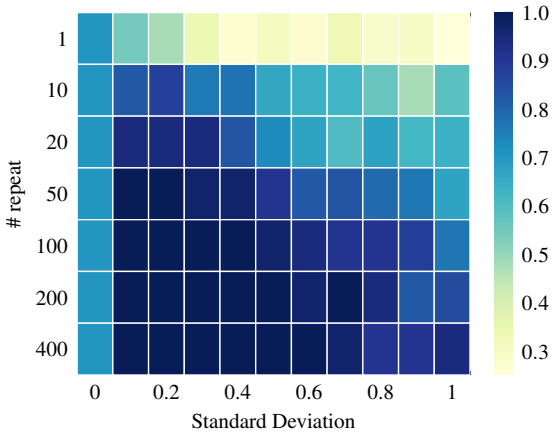


Fig. 4. Trade-off between the standard deviation of the Gaussian random variable in MAXPREP and the number or repeated heuristic runs of Algorithm 1. Values show the acceptance ratio compared to ILP$_a$, i.e., the higher is better.

our model is general enough to incorporate multiple revenue and cost models besides minimum latency to decide which hypervisor instance should be moved (if any), e.g., different Quality-of-Service classes, remaining request times, etc. In our simulations we follow a simple *conservative approach* where each request r is considered to be equal. However, we keep all existing vSDN requests in the network, while trying to maximize the acceptance ratio of novel vSDN requests in $\mathcal{R}$.

VI. Simulations

We investigated two topologies with similar sizes but different characteristics, namely the Janos-US (26 nodes, 42 edges, 3.23 average node degree) and Italy (25 nodes, 35 edges, 2.72 average node degree) networks [23]. We used multiple global latency requirements L given as the fraction of diameter. For each investigation, 10 independent simulations were performed, and in the figures, the average result of these simulations is shown. The vSDN requests are connected subgraphs in our network and the vSDN size in the figures refers to that size (i.e. $|\mathcal{V}^r|$). To evaluate the acceptance ratio, with each vSDN size ($2 \leq |\mathcal{V}^r| \leq |\mathcal{V}|$) 1000 (or maximum existing) unique vSDN request was generated.

We conducted 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. ILP instances are created with the Gurobi Python Interface (gurobipy) and solved with the Gurobi solver (version 9.5) [24].

A. Effect of Hypervisor Number on Acceptance Ratio

First, we analyzed the acceptance ratio with different number of active hypervisors k using the optimal ILP$_a$ method. One can observe in Figure 2 that with increasing number of hypervisors the acceptance ratio increases as well, but the effect of adding a new one decreases with more instances. An oblivious method like the Greedy heuristic in [6] which only tries to minimize the hypervisor number ($k = 6$ in the figure) might have poor performance in terms of acceptance ratio. However, we will demonstrate that even with the minimum number of hypervisors high-quality solutions can be achieved using an intelligent design through the MAXPREP function.

B. Intelligent versus Random Algorithm Design

Next, we compared Algorithm 1 to the Greedy heuristic approach [6]. We emphasize again that the Greedy heuristic finds only the minimum number of hypervisors k, and requires ILP$_a$ in a second step to maximize acceptance ratio, while our novel Algorithm 1 provides the result in a single step. For a fair comparison, we repeated both heuristics 400 times, and selected candidates both randomly and based on acceptance ratio described in Section IV-B. In Figure 3 one can observe that the Greedy approach with random candidate selection has the worst performance, while Algorithm 1 with the intelligent selection achieves 100% acceptance ratio for all investigated $\mathcal{R}_{rep}$ sets. Algorithm 1 outperforms the Greedy method even with random candidate selection when vSDNs are larger than four, which demonstrates that considering acceptance ratio through MAXPREP enables us to find high-quality solutions with most runs of Algorithm 1. It runs twice as fast as the two-step approach, and because less repeated runs are required to find a good solution, running time can be further reduced.

C. Required Number of Repeated Simulation Runs

Here we investigate the trade-off between the standard deviation of the Gaussian random variable used in MAXPREP and the number or repeated heuristic runs of Algorithm 1 to find a high-quality solution. One can imagine the effect of these two values as randomness increases the number of unique candidate hypervisor placements with size k we can select from, while the repeat number says how many times we try to pick a different hypervisor placement from this set. With a larger randomness we are able to explore more placements around the optimal solution to find one with

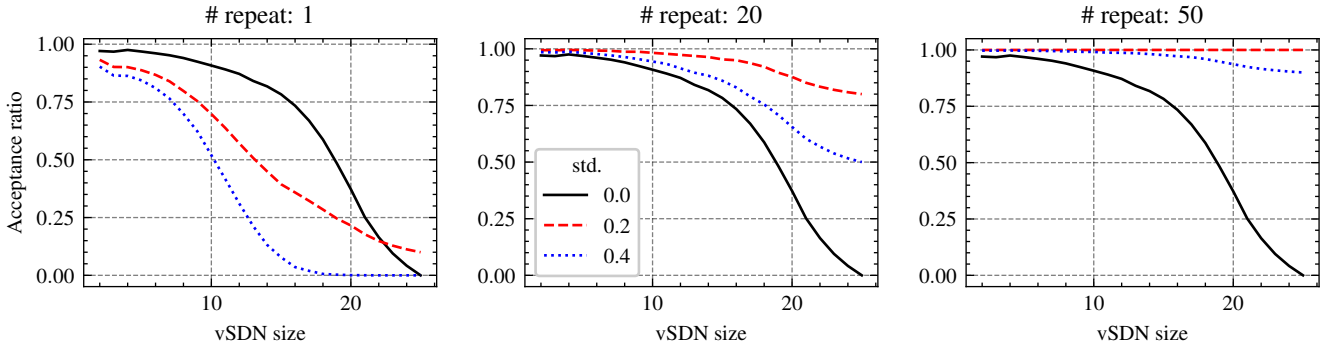


Fig. 5. Acceptance ratio in the Italy network with different number of repeated runs of Algorithm 1 and different standard deviations of the Gaussian random variable in MAXPREP. While repeat number significantly improves the results, the difference between parameters 0.2 and 0.4 are negligible.

maximum acceptance ratio for $\mathcal{R}_{rep}$. However, in a larger set we might need more tries to pick the best solution.

Figure 4 presents our results which compares the acceptance ratio of the found placements using Algorithm 1 to the optimal ILP$_a$ [6], i.e., 1.0 means that the optimal solution is found, while lower values mean weaker solutions. One can observe that above 50 simulation runs the results are not improving significantly, therefore we perform maximum 50 repeats in the rest of our simulations (instead of the 400 used for the Greedy method [6]). Furthermore, for the Gaussian random variable used in MAXPREP, the parameter values below 0.5 gives better performance than the larger ones.

We further investigated these parameter ranges in Figure 5. Although with a single run we were not able to observe any trend between different standard deviations, with more repeated runs the additional random factor significantly improved the acceptance ratio. We conclude that although randomness helps to find better solutions with extending the solution space of Algorithm 1, the effect of different variables and parameters is unclear on the performance and might change in different networks.

Reducing the number of repeats from 400 of the two-step approach to 50 in Algorithm 1 it becomes over 10 times faster on the investigated networks (reduces the control plane design process from 1150 sec to 87 sec even with an unoptimized Python code). The gap grows with increasing network size since the ILP of the two-step approach scales relatively bad.

D. Capacity Constraint on Hypervisors and Controllers

In order to understand the effect of the introduced constraints, we investigated the objective functions discussed in Section III. While we always considered Eq. (1) as the primary objective, in our first evaluation we changed the relative importance of Eq. (2) and Eq. (3) to each other. Considering the controller load as a secondary objective decreases it around 3% but the hypervisor load cannot be reduced significantly in this case. However, minimizing the maximum hypervisor load before the controller load, we can decrease it about 12% while still gain around 1% of controller load compared to the ILP$_a$ formulation in [6], i.e., when only Eq. (1) is considered. Therefore, we selected the latter one for further comparison.

Next, to fulfill strict requirements on the hypervisor loads, we have investigated the case when these constraints are incorporated into ILP$_a$:

$$\forall h \in \mathcal{H}^* : load(h) \leq max_{sw},$$

where max_{sw} is the upper bound on the number of switches managed by any hypervisor. In Figure 6 one can observe that the constraints have significant impact on the acceptance ratio, as with the upper bound the most popular hypervisor locations can serve only a given number of switches, thus, some vSDNs must be rejected. If such limitations must be considered, the network operator might need to increase the number of active hypervisors to achieve an acceptable performance.

E. Self-Adjusting Operation

Finally, we investigated the performance of our self-adjusting placement using the conservative approach, i.e., when in each timestep $t+1$ at most one hypervisor is migrated in order to maximize the acceptance ratio of the new $\mathcal{R}_{t+1}$, subject to the constraint that $\forall r \in \mathcal{R}_t \cap \mathcal{R}_{t+1}$ unfinished request must be accepted. Although the self-adjusting operation [13], [16] is advantageous for requests with high spatial locality (e.g., slow traffic load shifts as a day-night cycle), we wanted to evaluate the worst case scenario and selected vSDN requests into the request sets $\mathcal{R}_t$ uniformly random. Hence, in Figure 7 we simulated 20 independent runs with 400 consecutive timesteps each, and compared our resilient self-adjusting approach using at most one migration (denoted as 1) to the placement calculated with Algorithm 1 in each timestep ("Any" in the figure); and also to the method where we do not migrate at all (denoted as 0), i.e., where a single representative request set is used to find a high-quality initial placement with Algorithm 1 which is used for all 400 timesteps.

One can observe in Figure 7 that owing to the MAXPREP score the single placement for the first timestep (0 migration) with Algorithm 1 is already well-prepared and can serve 89% of all vSDNs. The self-adjusting operation (maximum 1 migration) outperforms it and achieves almost identical performance as the operation with "Any" number of migrations. This is in alignment with the table in the figure which shows the

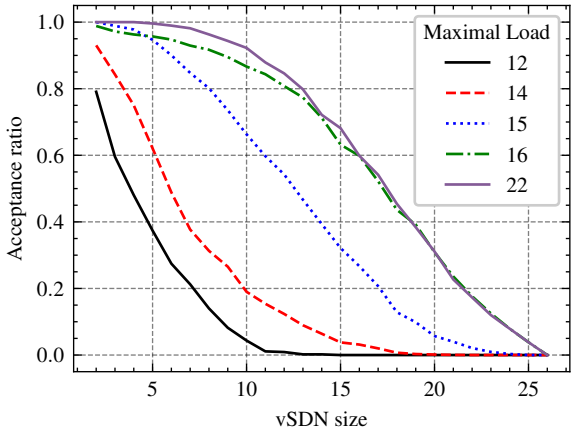


Fig. 6. The effect of different upper bounds on the number of switches served by each hypervisor on the acceptance ratio with 4 hypervisor instances in the Janos-US network ($L = 0.6$) calculated with the extended ILP$_a$ method [6].

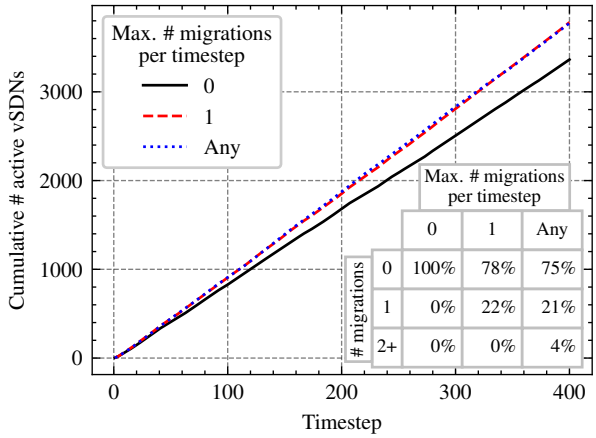


Fig. 7. Self-adjusting operation with different number of migrations available per timestep. The table shows the frequency of migration sizes.

frequencies of hypervisor migrations per timestep. In most cases only the migration of one hypervisor was sufficient to maximize the number of accepted vSDN requests and even though the unconstrained operation made 2 or more hypervisor migrations in 4% of the time, it did not improved its performance significantly. Therefore, we conclude that moving only a single hypervisor instance per timestep is desirable, as near-optimal performance can be achieved while service continuity is guaranteed for all requests.

VII. Conclusions

In this paper we demonstrated through the use case of resilient hypervisor placement that using Objective (i) and (ii) as the driving force of an intelligent algorithm design the acceptance ratio of vSDN requests can be improved, which leads to increased revenue for the network operator. On one hand, we proposed Algorithm 1 which considers possible future request and flexibility in the single-link and hypervisor failure resilient placement problem through a novel MAXPREP metric; thus, requires reconfiguration only if the request set has drastically changed. On the other hand, for dynamic environments we proposed a self-adjusting algorithm which adapts the hypervisor placement gradually for each new request set while continuously serving existing vSDNs in the network by using the same number of hypervisor instances. In the simulations we demonstrated that both of our algorithms provides well-prepared placements, and that our self-adjusting approach performs close to optimal even for the worst case (i.e., uniformly random) request changes.

Acknowledgments

This work was supported by Projects no. 134604 and no. 137698 that have been implemented with the support provided by the National Research, Development and Innovation Fund of Hungary, financed under the FK_20 and PD_21 funding schemes, respectively. The work of A. Pašić was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences. Project no. C1445813 has been implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed under the KDP-2021 funding scheme. The work of F. Mogyorósi and A. Pašić was supported by EMET under grant NTP-NFTÖ-22-B-0225 and NTP-NFTÖ-22-B-0141.

References

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

[2] A. S. Klyubin, D. Polani, and C. L. Nehaniv, "Empowerment: a universal agent-centric measure of control," in *2005 IEEE Congress on Evolutionary Computation*, vol. 1, Sep. 2005, pp. 128–135 Vol.1.

[3] C. Salge, C. Glackin, and D. Polani, *Empowerment–An Introduction*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 67–114.

[4] 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.

[5] P. Babarczi, "Resilient control plane design for virtual software defined networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 2557–2569, 2021.

[6] F. Mogyorósi, P. Babarczi, J. Zerwas, A. Blenk, and A. Pašić, "Resilient control plane design for virtualized 6G core networks," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 2453–2467, 2022.

[7] 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.

[8] M. He, A. M. Alba, A. Basta, A. Blenk, and W. Kellerer, "Flexibility in softwarized networks: Classifications and research challenges," *IEEE Comm. Surveys & Tutorials*, vol. 21, no. 3, pp. 2600–2636, 2019.

[9] P. Babarczi, M. Klügel, A. M. Alba, M. He, J. Zerwas, P. Kalmbach, A. Blenk, and W. Kellerer, "A mathematical framework for measuring network flexibility," *Computer Communications*, vol. 164, pp. 13–24, Dec 2020, Special Issue on IFIP Networking 2019 Conference.

[10] M. Tornatore, P. Babarczi, O. Ayoub *et al.*, "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. 1–24.

[11] H. Biallach, M. Bouhtou, D. Nace, and M. S. Mimouna, "An efficient heuristic for the virtual network function reconfiguration problem," in *2022 12th International Workshop on Resilient Networks Design and Modeling (RNDM)*, 2022, pp. 1–7.

[12] A. M. Alba, P. Babarczi, A. Blenk, M. He, P. Kalmbach, J. Zerwas, and W. Kellerer, "Modeling the cost of flexibility in communication networks," in *Proc. 40th IEEE International Conference on Computer Communications (INFOCOM)*, May 2021, pp. 1–10.

[13] B. Peres, O. Souza, O. Goussevskaia, S. Schmid, and C. Avin, "Distributed self-adjusting tree networks," in *Proc. IEE INFOCOM*, 2019.

[14] D. D. Sleator and R. E. Tarjan, "Self-adjusting binary search trees," *J. ACM*, vol. 32, no. 3, p. 652–686, jul 1985.

[15] ——, "Amortized efficiency of list update and paging rules," *Commun. ACM*, vol. 28, no. 2, p. 202–208, feb 1985.

[16] V. Addanki, M. Pacut, A. Pourdamghani, G. Retvari, S. Schmid, and J. Vanerio, "Self-adjusting partially ordered lists," in *IEEE INFOCOM*, 2023.

[17] 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.

[18] 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.

[19] 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.

[20] M. Pióro, M. Mycek, A. Tomaszewski, and A. d. Sousa, "On joint primary and backup controllers' placement optimization against node-targeted attacks," in *2022 12th International Workshop on Resilient Networks Design and Modeling (RNDM)*, 2022, pp. 1–7.

[21] 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.

[22] 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.

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

[24] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2021. [Online]. Available: https://www.gurobi.com