Fast and Efficient Network Service Embedding Method with Adaptive Offloading to the Edge

Balázs Németh[1], Márk Szalay[1], János Dóka[1], Matthias Rost[4], Stefan Schmid[5], László Toka[1,3], Balázs Sonkoly[1,2]
[1]Budapest University of Technology and Economics, [2]MTA-BME Network Softwarization Research Group,
[3]MTA-BME Information Systems Research Group, [4]Technische Universität Berlin, [5]University of Vienna
e-mails: {nemethb,szalay,doka.janos,toka,sonkoly}@tmit.bme.hu, mrost@inet.tu-berlin.de, stefan_schmid@univie.ac.at

***Abstract*—Next generation networks and applications have recently drawn the attention of many researchers in both academia and industry. The plethora of service orchestration, embedding and scheduling solutions being developed indicates that efficient resource utilization in the cloud and edge/fog architecture is of crucial importance in order to exploit the great economic potential of this infrastructure. In this paper we propose a novel service orchestration approach that aims for speed and quality in terms of service provisioning: our embedding algorithm instantly deploys end-to-end delay-constrained service graphs while regularly offloads the most burdened parts of the infrastructure applying a cost-aware VNF migration strategy. In this sense we propose a hybrid orchestration approach, which unites the advantages of online heuristic and offline optimizing service orchestration methods, with the goal of obtaining a system design that provides fast service placement decisions and efficient long-term operation. By an exhaustive evaluation of our orchestration system on core, cloud and edge network topologies, we show its benefits in efficiency and the collateral cost of migrations.**

I. Introduction

Clouds and cloud computing were the enablers of several novel applications and network services during the last decade. However, the "networking revolution" mainly driven by 5G and emerging applications with extreme requirements have posed new challenges on clouds and networked systems. On the one hand, strict latency bounds cannot be guaranteed between end devices and cloud resources in all scenarios due to geographic distribution. On the other hand, a number of novel services, such as IoT applications, require mass machine-to-machine communication transmitting a large amount of data between edge devices and the cloud. Fog computing is a promising solution recently proposed to address these issues. The key concept is to add compute resources to the edge of the network being closer to the customers or end devices in terms of latency. By these means, end devices can offload computational tasks to edge servers instead of using resources of data centers or their own. In addition, service elements traditionally running in the cloud can also be moved to the fog resulting in lower delays and in reduced network load.

In order to exploit the fog (cloud and edge) infrastructure efficiently, a novel resource orchestration design is indispensable. The orchestrator is responsible for computing the "best" placement of service elements meeting the given requirements, constraints or policies. More specifically, it assigns the Virtual Network Functions (VNFs) composing the service to compute resources (from the cloud or from available edge domains) and also allocates paths between connected VNFs. As future systems should cope with a large number of incoming requests and the underlying infrastructure topologies can be complex, the performance characteristics of the mapping (or embedding) process is crucial with respect to runtime properties and the quality of embeddings it calculates. Furthermore, the requests arrive and are deployed one after another which can result in a sub-optimal global system state after a long time of operation without further considerations. To mitigate this problem, periodical re-optimization is needed. More precisely, the orchestrator recalculates the placement of each (or only selected) service components whenever certain conditions are met, thus triggering a partial redeployment. In fact, the corresponding reconfigurations require the migration of certain VNFs and updating routes of network paths, hence posing challenges when intending to avoid service disruptions. The cost of the migrations highly depends on the service characteristics (stateless or stateful VNF) and the underlying technologies.

In this paper, we propose a novel resource orchestration mechanism which can efficiently use compute and network resources in emerging fog computing environments. The core of the orchestrator is our hybrid embedding algorithm that combines the benefits of an online heuristic method and an offline, ILP-based optimization approach. We evaluate our algorithm in several network scenarios and topologies from different aspects, including the impact of offloading, i.e., VNF migrations. Our analysis reveals significant advantages of our design in certain scenarios and a behavior similar to the online approach in other special cases.

The rest of the paper is organized as follows. In Sec. II, a brief summary on the related work is given. We present our service and resource models and the service embedding problem in Sec. III. Sec. IV is devoted to our orchestration algorithms. A detailed numerical evaluation of the proposed methods is presented in Sec. V. Sec. VI draws the conclusion.

II. Related Work

Placing services in the cloud and in the fog or edge is analogous with the Virtual Network Embedding (VNE) problem that has been studied intensively in the last decade. As it is known to be NP-hard [1], finding the optimal solution within reasonable time is not feasible in case of realistically sized infrastructure topologies and complex service requests. The body of research around VNE is vast: many papers present solutions either with exact optimum for limited scale scenarios,

or close-to-optimal results achieved by heuristic algorithms. A summary of solutions to the VNE problem is given in [2].

Selected research works that belong to the first group are [3] and [4]: in the former the authors use Integer Linear Programming (ILP) to solve the VNE problem for minimizing the cost of embedding in terms of edge costs while maximizing the acceptance ratio; in the latter the authors propose a MIP (Mixed ILP) formulation for reconfiguring existing mapping by enabling migrations. The second group, i.e., approaches that solve the VNE problem with heuristic algorithms, also shows a colorful palette of ideas. The authors of [5] propose a hybrid algorithm, which first solves the LP problem that is a relaxed version of the original ILP, then use deterministic and randomized rounding techniques on the solution to approximate the results of the original MIP. Other works perform the mapping in two steps: a node and an edge mapping stage. A decomposing mapping algorithm proposed in [6] aims to minimize the mapping cost by making a selection of the available decompositions during the node mapping stage.

Another dimension along which the related works can be grouped is whether reconfigurations are considered or not. One of the first works that applied reconfigurations in online embedding algorithms is [7]. Every time the embedding of a service fails, a greedy reconfiguration algorithm tries to migrate one virtual node with its incident virtual edges to make room for the failed request. Our work differs from this one, because optimizations are run in the background and considers reconfiguration of several request graphs at once. In the realm of embedding virtual clusters in data centers, migrations of virtual machines were also studied to enable *upgrades* of already deployed services by Fuerst et al. [8]: given the specific star-shape structure of the virtual cluster requests, the authors gave an algorithm that finds the minimal resource embedding under any number of allowed migrations.

Authors of [4] argue that re-embeddings can be useful if the deployment of services were requested at short notice and initially placed heuristically. However, the authors only present super polynomial-time algorithms: they describe a MIP to compute optimal embeddings for different objective functions with cost-aware migrations. A more focused work [9] tackles the embedding of virtual data centers into physical data centers, also allowing for migrations, and specifically maximizing the total revenue, while minimizing the total energy consumption. In our design we aim for a more flexible algorithm that can be widely customized.

Standardization organizations provide reference architectures and their respective implementations, which provide frameworks for real life deployments of such orchestration systems, one example is ETSI's Open Source MANO [10].

To the best of our knowledge, we are the first to propose and evaluate an orchestration system which composes self-contained algorithms of different approaches on a service and resource model that allows for delay-constrained specification in contrast to the general VNE model.

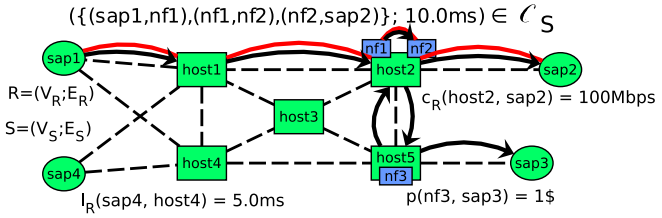


Fig. 1. An example for the Service Graph Embedding problem.

III. Model and Problem Definition

A. Service and Resource Model

We use the "Big Switch with Big Software" (BiSBiS) model to represent the physical network, which was originally introduced by the EU-funded FP7 UNIFY project(http://fp7-unify.eu). Formally, the network is represented by the *resource graph* $R = (V_R, E_R)$. A single node in this network may represent a single physical server or a whole data center. We denote all network function (NF) types by $\mathcal{T}$, e.g. firewall, deep packet inspector, general purpose server are depicted by $\mathcal{T} = \{\text{FW}, \text{DPI}, \text{x86-server}\}$ respectively. Additionally, we assume that $\mathcal{T}$ includes type identifiers for Service Access Points (SAPs), i.e., resource nodes at which flow may enter or leave the network, such that only the node $\text{SAP}_k \in V_R$ supports the type $\text{SAP}_k \in \mathcal{T}$.

Furthermore, considering an arbitrary set of node resources $\mathcal{R}$, e.g. $\mathcal{R} = \{\text{CPU}, \text{RAM}, \text{Disk}\}$, each node $u \in V_R$ offers a specific capacity $c_R(u, r)$ of each resource $r \in \mathcal{R}$. Each abstract link $(u, v) \in E_R$ is attributed with its bandwidth $c_R(u, v)$ and its latency $l_R(u, v)$ (cf. Fig. 1).

Services are similarly described by a graph abstraction, namely the *service graph* $S = (V_S, E_S)$. Each node $i \in V_S$ is attributed a specific function type $\tau_S(i) \in \mathcal{T}$ with specific resource demands. Mapping the function i on a resource node u requires $d_S(i, u, r)$ many 'units' of resource r. Any edge $(i, j) \in E_S$ of the service graph is attributed with a bandwidth requirement $d_S(i, j)$. Furthermore, latency constraints can be defined for services along *chains*. Each service comes with a set $\mathcal{C}_S$ of tuples (c_p, l_p), where c_p denotes a path in the service graph and l_p denotes the maximal allowed latency on this path. Notations are summarized in Tab. I.

B. The Service Embedding Problem

An illustrative example of the service graph embedding problem is shown in Fig. 1. A possible solution of a simple

TABLE I
Mathematical notations used in this paper.

Notation	Description
V_G, E_G	Vertices and edges of graph G
$\mathcal{P}_G \subseteq \mathcal{P}(E_G)$	Simple paths of graph G
$\mathcal{T}$, $\mathcal{R}$	Set of NF and resource types
$\tau_R : V_R \mapsto \mathbb{P}(\mathcal{T})$	Supported NF types
$\tau_S : V_S \mapsto \mathcal{T}$	Function type of an NF
$d_S : V_S \times V_R \times \mathcal{R} \cup E_S \mapsto \mathbb{R}_{\geq 0}$	Resource demands of the service graph (both nodes and links)
$c_R : V_R \times \mathcal{R} \cup E_R \mapsto \mathbb{R}_{\geq 0}$	Capacities of resource graph
$l_R : E_R \mapsto \mathbb{R}_{\geq 0}$	Delay of links
$\mathcal{C}_S \subseteq \mathcal{P}_S \times \mathbb{R}_{\geq 0}$	Path latency constraints of services
$p : V_S \times V_R \cup E_S \mapsto \mathbb{R}_{\geq 0}$	Cost (price) of mapping nodes and of using bandwidth

service graph mapping onto the network of resource nodes is depicted with a given path requirement between two SAPs. Each NF must be mapped to a single resource node and each service graph link must be mapped to a simple path of the substrate network. A feasible embedding must respect the semantics, capacity requirements and QoS constraints of the service graph. A mathematically exact formulation of the problem is explained in Sec. IV-A.

IV. Our orchestration algorithms

Given the informal problem statement above, we first introduce the Mixed-Integer Programs (MIP) used for (i) reconfiguring existing embeddings and (ii) computing optimal offline solutions. Then we present the greedy heuristics used for serving requests in an online fashion, third we propose the hybrid algorithm that incorporates (optimal) reconfigurations. The source code of the presented algorithms are available (https://github.com/hsnlab/mapping).

A. Mixed-Integer Program for Offline Optimization

We first explain how the Mixed-Integer Programming formulation computes embeddings (cf. MIP 1 and Tab. II) and afterwards present the objective of the optimization.

As discussed above, the task is to find a mapping of the service graph $S = (V_S, E_S)$ to the resource graph $R = (V_R, E_R)$. To represent the node and link mappings, binary variables $x_u^i \in 0, 1$ and $y_{u,v}^{i,j} \in 0, 1$ are employed: $x_u^i = 1$ indicates that service node i is mapped on resource node u and $y_{u,v}^{i,j} = 1$ indicates that the resource link $(u, v) \in E_R$ is used to establish the service link $(i, j) \in E_S$.

Mixed-Integer Program 1: Service Embeddings

$$\min \left(\begin{array}{ll} \alpha \cdot \sum_{(u,v)\in E_R} a_{u,v} \cdot p(u,v) & + \\ \beta \cdot \sum_{i\in V_S, u\in V_R} x_u^i \cdot p(i,u) & + \\ \gamma \cdot (1 - U_{V_R}^{\min}) & \end{array} \right) \tag{1}$$

$$\sum_{u\in V_R, \tau_S(i)\in\tau_R(u)} x_u^i = 1 \qquad \forall i \in V_S \tag{2}$$

$$\sum_{u\in V_R, \tau_S(i)\notin\tau_R(u)} x_u^i = 0 \qquad \forall i \in V_S \tag{3}$$

$$\sum_{(u,v)\in\delta_u^+} y_{u,v}^{i,j} - \sum_{(v,u)\in\delta_u^-} y_{v,u}^{i,j} = x_u^i - x_u^j \qquad \forall (i,j) \in E_S, u \in V_R \tag{4}$$

$$\sum_{i\in V_S} x_u^i \cdot d_S(i,u,r) = a_{u,r} \qquad \forall u \in V_R, r \in \mathcal{R} \tag{5}$$

$$\sum_{(i,j)\in E_S} y_{u,v}^{i,j} \cdot d_S(i,j) = a_{u,v} \qquad \forall (u,v) \in E_R \tag{6}$$

$$a_{u,r}/c_R(u,r) \geq U_{V_R}^{\min} \qquad \forall u \in V_R, r \in \mathcal{R} \tag{7}$$

$$a_{u,r} \leq c_R(u,r) \qquad \forall u \in V_R, r \in \mathcal{R} \tag{8}$$

$$a_{u,v} \leq c_R(u,v) \qquad \forall (u,v) \in E_R \tag{9}$$

$$\sum_{(i,j)\in c_p, (u,v)\in E_R} y_{u,v}^{i,j} \cdot l_R(u,v) \leq l_p \qquad \forall (c_p, l_p) \in \mathcal{C}_S \tag{10}$$

TABLE II
Mathematical notations used in MIP formulation.

$x_u^i \in \{0,1\}$	Variable indicating mapping of $i \in V_S$ on $u \in V_R$
$y_{u,v}^{i,j} \in \{0,1\}$	Variable indicating the mapping of $(i,j) \in E_S$ on $(i,j) \in E_S$
$a_{u,r} \in \mathbb{R}_{\geq 0}$	Variable equaling total node allocations for $u \in V_R$ and $r \in \mathcal{R}$
$a_{u,v} \in \mathbb{R}_{\geq 0}$	Variable equaling total bandwidth allocations on $(u,v) \in E_R$
$U_{V_R}^{\min} \in [0,1]$	Variable indicating the minimum node utilization
α, β, γ	Coefficients for normalizing and weighting objective function
$p_{i,u} \in \mathbb{R}_{\geq 0}$	Price for placing service node $i \in V_S$ on resource node $u \in V_R$
$p_{u,v} \in \mathbb{R}_{\geq 0}$	Price for using bandwidth along resource edge $(u,v) \in E_R$

Constraints (2) and (3) enforce that each service graph node is mapped onto a suitable resource graph node while forbidding mappings to nodes that do not support the respective function type. We note that when a service node is attributed with the type SAP_k, these constraint force this node to be mapped on the respective resource node $\text{SAP}_k \in V_R$.

Constraint (4) induces a unit-flow for each service link $(i, j) \in E_S$ using the flow variables $y_{u,v}^{i,j} \in \{0, 1\}$ for all resource edges $(u, v) \in E_R$. The left-hand side of Constraint (4) states flow preservation, while the right-hand side enforces the sending of a unit flow from the node onto which the tail node i is mapped while the node onto which the head j is mapped must receive a unit of flow. Note that, in the case that both i and j are mapped to the same node $u \in V_R$, then no network path needs to be established in the resource graph.

Constraints (5) and (6) compute the allocations induced by the node and link mapping, respectively, and Constraints (8) and (9) enforce that these allocations are upper bounded by the capacities of the respective resource graph elements. In particular, considering a resource node $u \in V_R$, Constraint (5) sums up all the allocations induced by the mapping of service nodes to u, yielding the variable $a_{u,r} \geq 0$ for all resources $r \in \mathcal{R}$. In turn, Constraint (8) bounds this (total) allocation by the respective capacity $c_R(u, r)$. Capacity constraints for the link mappings are enforced in the similar fashion using Constraints (6) and (9) while only a single resource, namely the bandwidth, is considered.

To enforce latency constraints for the set of chains $\mathcal{C}_S$ Constraint (10) is used. For each tuple $(c_p, l_p) \in \mathcal{C}_S$, the sum of delays of all resource edges used by any of the service links is computed (left-hand side) and is upper bounded by the maximum allowed latency l_p (right-hand side).

Having described how MIP 1 computes *valid* embeddings of services, we now turn to its objective. The first two summands of the objective describe costs for resource allocations on edges and nodes respectively, while the third is used for load balancing. Each of these summands is multiplied by a specific scaling factor $\alpha, \beta, \gamma \geq 0$, which is used to (i) normalize and (ii) weight the different components of the objective. The first summand expresses costs for using bandwidth by employing prices $p(u, v) \geq 0$ which can be set by the provider according to the importance of the respective links. The second summand expresses costs for mapping a service node $i \in V_S$ onto a resource node $u \in V_R$ using prices $p(i, u) \geq 0$. In particular, migration costs of functions can be modeled by setting $p(i, u) = 0$, if i was previously mapped on node u,

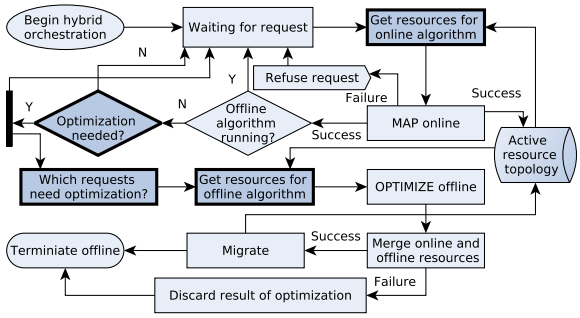


Fig. 2. Operation of the hybrid algorithm with the strategic decision points.

and setting $p(i,u) > 0$ if this was not the case.

Lastly, for the load balancing summand, we use one additional variable $U_{V_R}^{\min} \geq 0$, which shall denote the minimum (node) resource load among all resource nodes and resource types. Constraint (7) upper bounds this variable by the allocations $a_{u,r}$ of resource $r \in \mathcal{R}$ on node $u \in V_R$ divided by the respective capacity. Hence, $U_{V_R}^{\min}$ must be less than the (relative) load with respect to any node and resource. Hence, by minimizing $(1 - U_{V_R}^{\min})$, the minimum load shall be increased, leading to distributing load more evenly.

B. Heuristic Online Embedding Algorithm

Our online heuristic algorithm is designed for embedding service graphs with delay, bandwidth and cost requirements efficiently. A detailed description and evaluation of the algorithm can be found in [11]. The heuristic mapping algorithm searches for possible embedding of service chains, considering a greedy step as the combined mapping of a network function and an adjacent service graph link. The greedy mapping is guided by parameterizable preference metrics to identify the locally best steps. If the greedy search fails, a bounded backtracking procedure is responsible for exploring a subset of the state space by trying locally less preferred steps. The preference metrics consider load balancing on nodes, minimizing link resource usage, and they guide the mapping process to terminate as early as possible while complying to end-to-end delay requirements. Easily controllable preference values enable to use the algorithm in various network environments. For example, in [12] we adopt the algorithm to implement a data plane orchestrator deploying service graphs directly on low-level hardware resources.

C. The Proposed Hybrid Algorithm

The idea of our hybrid orchestration algorithm is to mix the advantages of the heuristic and the MIP-based algorithms, so that the response time of the orchestration system would stay low by quickly finding a resource allocation for the service graph, while the network resource utilization is kept efficient, provided by the reconfigurations. Our proposed hybrid orchestration algorithm uses the previously explained algorithms in parallel: (i) the heuristic algorithm explained in Sec. IV-B is used to map the incoming service graphs in an *online* manner as they arrive, (ii) the MIP-based optimization presented in Sec. IV-A is used to occasionally reconfigure the resource reservation of a larger set of services in an *offline* manner.

The hybrid operation is shown by Fig. 2. As a new service request arrives, the state of the current resource topology is gathered, which can be used by the online algorithm for mapping the service graph immediately. If the orchestration algorithm terminates with failure, the service request is refused, while in case of success the active resource topology is updated. When an offline optimization is not in progress, a new optimization can be initiated after any successful online embedding cycle. During any phase of the offline optimization, the new incoming requests are handled by the online orchestration loop. Before starting the MIP solver, the set of already mapped service graphs to be optimized must be selected and a (possibly partial) view of the current resource topology must be given to the offline orchestration algorithm. When the optimization succeeds, its result must be merged with the online algorithm's current view of resource allocations, where the new requests arrived during the offline operation are incorporated. This is necessary for applying the optimization result on the active resource topology. The merge may not be possible if the reservation would exceed resource capacity on any network element, so the optimization result must be discarded. In case of a successful merge, the optimization result is incorporated by applying the suggested migrations, and the active resource topology is updated.

Strategic decision points: The emphasized boxes in Fig. 2 are customizable parts of the algorithm which can be used to tune the hybrid orchestrator for its application environment. *Resource division strategies* allow the network operator to control how much resource is allocated to the online and offline algorithms during an optimization cycle, respectively. This strategy controls the prevalence of merge failures after an optimization has terminated. Running an optimization and migrating the service graphs are expensive in terms of computation resources and service instability, so we need to control when to start an offline optimization procedure; this is enabled by the *optimization interval* strategies. The *requests to be optimized* strategy selects a set of already mapped service graphs for the offline optimization. A more detailed explanation of the strategies can be found in [13].

Migration costs: The offline optimization must consider the cost of NF migration. When an offline optimization is initiated, an allocation is available for all, already mapped service graphs, so each NF has a single hosting resource graph node. Based on the current reservation, the migration cost can be defined by the function $p(i,u)$, so the β component of the MIP's objective function (1) expresses the total migration cost of the whole optimization. The migration cost may take into account any paramter of the VNF or the affected service.

V. Numerical evaluation

A. Methodology

Firstly, for evaluation purposes, we have defined a benchmark service graph sequence consisting of 1000 generated non-branching paths of 1 to 8 NFs connecting two randomly chosen SAPs of the resource graph $R = (V_R, E_R)$. We denote the respective sequences as Σ_R shown in (11). The functional

TABLE III
GRAPH PARAMETERS USED DURING THE SIMULATIONS.

Param.	Request	*gwin*	*edge_and_core*	*spine_leaf*
Nodes	Unif.(1,8)	23	25	37
SAPs	2 rand.	20	20	20
Links	chain	158	162	130
BW	5	$10^2, 10^3, 10^4$	$10^2, 10^3, 10^4$	10^4
CPU	2	6×400	6×267 and 2×400	6×400
NF types	10	6 each	6 each	6 each

type of each NF is independently chosen from a set of 10 abstract NF types. The end-to-end latency requirement of each service graph path depends on the resource graph R's capabilities.

$$\Sigma_R = \left\{ \left(V_S^i, E_S^i, \mathcal{C}_S^{R,i}, X_i, Y_i^R \right) \right\}_{i \in \{1,\dots,1000\}} \tag{11}$$

The inter-arrival times X_i of the service graphs are exponentially distributed $X_i \overset{iid}{\sim} Exp\left(\frac{1}{10}\right)$ in all cases, while the exponentially distributed lifetimes Y_i^R may be different for the resource graphs.

For tractability reasons of the required number of simulations, the other parameters (e.g. bandwidth, CPU requirement) are not randomized, their fixed values are summarized in Tab. III. In all of the simulation setups, the CPU is the bottleneck among the node resources, so only this resource is considered in the remainder of the paper.

Secondly, we have evaluated our hybrid orchestration system on three different resource topologies, which have the same amount of total node resources.

1) gwin: a German backbone topology taken from SNDlib (http://sndlib.zib.de) is extended with 6 access switches, 6 computation nodes and 20 SAPs, summing up to 43 nodes and 158 links. Link bandwidths are grouped as access, aggregation and core links with ascending values as shown in Tab. III.

2) edge_and_core: a modified version of *gwin*, where every access switch has its own edge computing node and 2 nodes are left connected to the core switches, summing up to 45 nodes and 162 links. The number of CPUs varies in core and edge computation nodes as shown in Tab. III.

3) spine_leaf: a cloud computing network topology with 31 switches, 6 computation nodes and 20 SAPs interconnected by 130 links in a spine-leaf network [14]. Each computation node supports 6 NF-types, randomly chosen from the set of 10.

Furthermore, in order to provide a reasonable comparison of the performance of our orchestration systems on different topologies, the difficulty of embedding a service graph sequence should be equal on each resource network topology.

Definition 1. *$\mathcal{D}(\Sigma_R)$ is the ratio of the number of refused service graphs and the number of total service graphs of the sequence Σ_R orchestrated over resource graph (V_R, E_R) using the MIP-based embedding algorithm while utilizing 90% of the CPUs of the whole network in 90% of the time.*

We have defined a service graph sequence for all three resource topologies, so that their difficulty would be approximately identical with a refusal rate of 10%, i.e. $\mathcal{D}(\Sigma_{gwin}) = \mathcal{D}(\Sigma_{edge\_and\_core}) = \mathcal{D}(\Sigma_{spine\_leaf}) = 0.1$. The reasoning behind choosing this definition and difficulty value are (i) on the one hand, too low or too high resource utilization may show no difference in algorithm performance, because they lead to easily solvable or impossible embedding tasks, (ii) on the other hand, we would like to avoid making capacity measurement-like benchmarking, when only the utilization of resources would influence algorithm performance, so the difficulty shall be affected by service graph constraints.

The method of defining a service graph sequence Σ_R with given difficulty according to our definition using delay constraints and service lifetimes is detailed in [13].

B. Hybrid Algorithm Settings

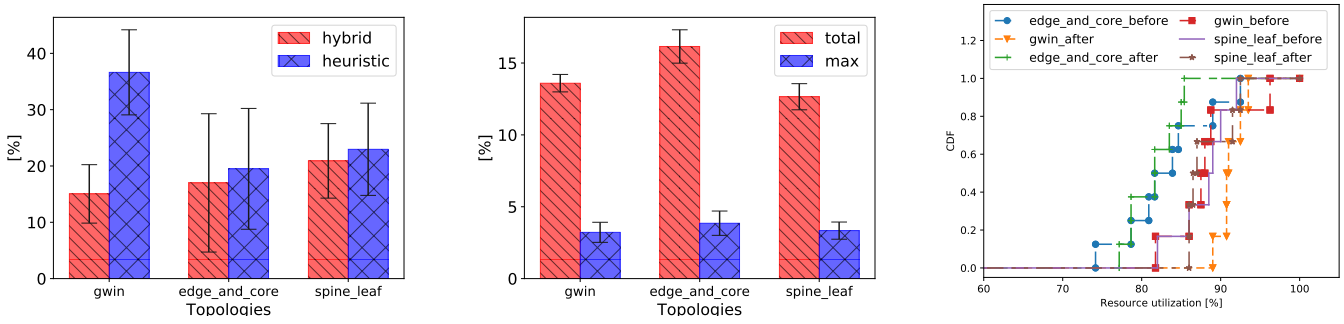
We have set the strategic decision points of the hybrid algorithm identically for all test cases. For the resource division strategy both online and offline algorithms are offered all available capacities, realizing a bold approach. A fixed arriving service graph counter is chosen for the optimization trigger, which initiates the offline algorithm after 10 arriving requests. During the offline operations, *all* of the service graphs present in the system are considered for optimization. We defined a constant migration cost function, which returns a fixed, high value if a NF needs to be moved independently from the target host of migration, and zero cost if the NF stays on the same resource graph node according to the optimization result.

At last, for simulating the parallel operations of the online orchestration cycles and the offline optimization, we have introduced a delay of applying the optimization result on the data structure of the hybrid orchestrator's active resource topology. On the one hand, this time interval simulates the running time of the offline algorithm, on the other hand, it can be accounted for any reconfiguration task required in a real-life system, such as NF state migration, traffic rerouting, etc. After the application delay of 10 arriving service graphs expires and they are greedily mapped, the resource views are merged, as explained earlier in Sec. IV-C.

C. Simulation Results

We evaluated our algorithms on the presented topologies $R \in \{$*gwin*, *edge_and_core*, *spine_leaf*$\}$ by considering 10 generated service graph sequences Σ_R for each topology according to the parameters shown in Tab. III with equal difficulties $\mathcal{D}(\Sigma_R) = 0.1$. All service graph sequences were mapped by all three algorithms under the same circumstances: each resource graph starts from an equally loaded state close to its capacity to simulate long-term operation of the orchestration system. Results are shown in Fig. 3.

Firstly, the MIP-based algorithm refuses around 10% of service graphs in each generated sequence Σ_R by design of the difficulty setting of each sequence. Fig. 3a shows how many service graphs the other algorithms refused additionally proportional to the MIP-based algorithm's refusal count. The error bars show the standard deviations of the sample. The biggest improvement is achieved by the hybrid algorithm on topology *gwin*, where it reduces the additionally refused number of service graphs by 22% compared to the heuristic



(a) (b) (c)

Fig. 3. Simulation results: (a) Performance comparison of the different algorithms based on the number of additionally refused requests compared to the MIP-based algorithm's performance on different topologies. (b) Average total migrated NFs during the service graph sequences; and the average of the maximal migrated NFs per optimization. The values are compared to the total number of NFs of a sequence and the present NFs in the resource graph respectively. (c) Cumulative distribution functions of CPU resource utilizations before and after an optimization of the hybrid algorithm on all three topologies.

algorithm's performance. The primary reason for the lower gain on the other topologies is the good relative performance of the heuristic mappings, which is only around 20% worse than the MIP-based algorithm. This is the result of the evenly distributed computation nodes between any two SAPs of the *edge_and_core* and *spine_leaf* topologies, where the heuristic algorithm performs well.

Secondly, we have examined the number of NF migrations required for the shown performance improvements of the hybrid orchestrator on all resource graph topologies. Fig. 3b shows the ratios of total NF migrations happened during the hybrid operation proportional to the number of NFs mapped during the whole service graph sequence, furthermore, it shows the maximal number of NF migrations necessary in a single reconfiguration step of an orchestration sequence proportional to the amount of NFs currently present in the network. According to our results, less than $1/20$ of the NFs are affected by an optimization and it is only around 15% of the NFs which were ever mapped successfully to the resource graph had to be moved at least once.

At last, we analyze how CPU resource utilization of computation nodes change over time on all three resource graph topologies during the hybrid orchestration. Fig. 3c shows the cumulative distribution functions of CPU utilizations of the resource graphs in hand-picked moments before and after applying the optimization results. For instance, the CPU utilizations are ranging from 81% to 97% before the optimization, while this interval mitigates to 89% to 93% after the successful migrations on the resource graph *gwin*. As the effect of the offline algorithm's objective, the least used computation node has higher utilization while the most loaded node has lower utilization after the optimization in all examples.

VI. Conclusion

Orchestration systems of next generation networks need to consider VNF migration possibilities to mitigate the problem of sub-optimality of sequential service graph embedding. We formulated the service graph embedding problem as a mixed integer program, and we introduced a novel orchestration approach mixing the advantages of a heuristic online embedding algorithm and the MIP-based optimum in form of our hybrid orchestrator. We proposed our interpretation of service graph sequence difficulty, which we have used to evaluate our novel orchestration approach on various topologies. Our results show that our hybrid orchestrator yields an improvement in deployments of up to 20% on core networks compared to the online approach. Furthermore, we demonstrated the robustness of edge computing and cloud computing topologies for next generation networks. Finally, we emphasized the importance of having computation nodes distributed on the end-to-end paths for an easy compliance with end-to-end delay requirements.

Acknowledgement

This research was supported by H2020-ICT-2014 project 5GEx (grant agreement no. 671636), which is partially funded by the European Commission. Project no. PD 121201 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the PD_16 funding scheme. This work was partially supported by the German BMBF Software Campus grant 01IS1205.

References

[1] E. Amaldi *et al.*, "On the computational complexity of the virtual network embedding problem," *Electronic Notes in Discrete Mathematics*, 2016.
[2] A. Fischer *et al.*, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, 2013.
[3] I. Houidi *et al.*, "Virtual network provisioning across multiple substrate networks," *ComNet*, 2011.
[4] G. Schaffrath *et al.*, "Optimizing long-lived cloudnets with migrations," in *IEEE/ACM UCC*, 2012.
[5] M. Chowdhury *et al.*, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM ToN*, 2012.
[6] S. Sahhaf *et al.*, "Network service chaining with optimized network function embedding supporting service decompositions," *ComNet*, 2015.
[7] I. Fajjari *et al.*, "VNR algorithm: A greedy approach for virtual networks reconfigurations," in *IEEE GLOBECOM*, 2011.
[8] C. Fuerst *et al.*, "Kraken: Online and elastic resource reservations for multi-tenant datacenters," in *IEEE INFOCOM*, 2016.
[9] M. F. Zhani *et al.*, "VDC planner: Dynamic migration-aware virtual data center embedding for clouds," in *IFIP/IEEE IM*, 2013.
[10] (Oct. 2017). Osm release three – a technical overview, ETSI OSM Community, [Online]. Available: https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseTHREE-FINAL.PDF (visited on 02/2018).
[11] B. Németh *et al.*, "Efficient Service Graph Embedding: A Practical Approach," in *IEEE NFVSDN O4SDI*, 2016.
[12] B. Sonkoly *et al.*, "FERO: Fast and Efficient Resource Orchestrator for a Data Plane Built on Docker and DPDK," in *IEEE INFOCOM*, 2018.
[13] B. Németh *et al.*, "Hybrid resource orchestration algorithms," BME, Tech. Rep., Jan. 2018. [Online]. Available: https://sb.tmit.bme.hu/techrep-hybrid-2018.pdf.
[14] M. Alizadeh *et al.*, "On the data path performance of leaf-spine datacenter fabrics.," in *IEEE Hot Interconnects*, 2013.