Cost-Efficient Resource Allocation Method for Heterogeneous Cloud Environments

Marton Szabo[1], David Hajay[2] and Mark Szalayz[3]

***Abstract*—In this paper we present a novel on-line NFV (Network Function Virtualization) orchestration algorithm for edge computing infrastructure providers that operate in a heterogeneous cloud environment. The goal of our algorithm is to minimize the usage of computing resources which are offered by a public cloud provider (e.g., Amazon Web Services), while fulfilling the required networking related constraints (latency, bandwidth) of the services to be deployed. We propose a reference network architecture which acts as a test environment for the evaluation of our algorithm. During the measurements, we compare our results to the optimal solution provided by an ILP-based solver.**

***Index Terms*—orchestration; network algorithm; heterogeneous cloud, fog computing, cloud computing**

I. Introduction

In the field of telecommunications many new emerging trends can be observed. For example, IoT (Internet of Things) aims to make traditional devices smart and connected to the Internet; this is a major trend already present nowadays. In the field of transportation, we can also find many exciting new solutions (e.g., remote driving, autonomous drones, etc.). The appearance of 5G networks is expected to enable even more revolutionary services to be built [1]. These can be for example the tactile Internet and on-line augmented reality applications, where the low response time is a crucial prerequisite. These services require not only the evolution of the radio interface, but also necessitates certain modifications in the topology of the back-haul network in order to serve the large number of new devices and the network traffic generated by them, and provide near real-time response times.

Today's widely deployed telecommunications networks are not flexible enough to fulfill these expected new challenges. For example, running network functions are currently binded to the special purpose hardware elements located in the core of the network (e.g. firewalls, carrier grade NAT platforms), which means unbearably high latency for most of the new applications. The NFV (Network Function Virtualization) concept aims to overcome this challenge [2], [3]: virtualization of the network functions makes it possible to run these services on general purpose hardware (e.g., x86 based servers with high compute capacity), thus removing the limitations coming from the physical location of the devices. The virtual network functions are expected to be one the fundamental building blocks of the future 5G networks.

Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics, Hungary, 1117 Budapest, Magyar Tudosok krt. 2.

[1] szabo.marton@tmit.bme.hu, [2] yhaja.david@tmit.bme.hu,
[3] zmark.szalay@tmit.bme.hu

DOI: 10.36244/ICJ.2018.1.3

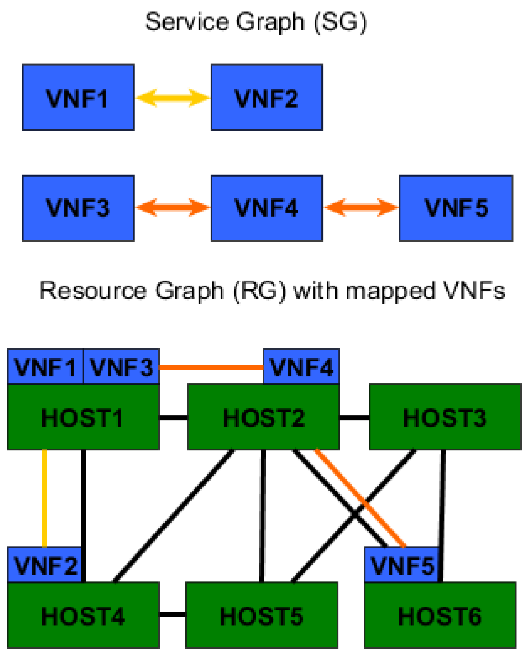

Figure 1. Mapping Service Function Chain to the infrastructure

Furthermore, by extending the traditional cloud concept with compute nodes at edge of the network – often called Mobile Edge Computing (MEC) [4] – using together with the high amount of resources in the core data centers enables many new applications for the service providers. This way, it is possible to run certain network functions near to the end-users with very low latency guaranteed, while other components of a service – that are not so sensitive to latency – can be deployed in core data centers instead of placing those in the limited capacity edge nodes [5]. One service consists of elementary functions connected with each other in a given order. This is called Service Function Chaining (SFC), and its model defines different requirements to the underlying network and virtualization environments (required CPU, RAM, storage, constraints on bandwidth, latency between nodes) [6]. The process that maps multiple service graphs (SGs) composed of different virtual network functions (VNFs) to a common physical infrastructure, represented by the Resource Graph (RG), is called Virtual Network Embedding (VNE). An example of the placement of VNFs and logical connections to the nodes and links of the physical infrastructure is shown in Fig. 1.

By applying the previously described technologies together with dynamically reconfigurable, software-based networks (Software Defined Networks, SDN), limitations caused by the current rigid network architectures can be eliminated, thus making the introduction of the new-generation network services possible. These can be for example remote driving cars and industrial robots controlled from the cloud, edge content caching, smart cities or on-line augmented reality applications.

An other interesting aspect of the future 5G networks is the resource sharing between different service providers, which would enable their users to be served independently of their actual physical location (e.g., in case of roaming). In such a multi-provider cloud environment the goal of the participants is to utilize their own infrastructure the most efficiently, thus minimizing the expenses caused by using external resources.

In this paper, we propose a new online resource orchestration algorithm which finds proper placement for the network functions of online services while minimizing the costs to be paid for external resources taken at third-party infrastructure providers. In order to evaluate our algorithm, we implemented a framework where we tested its performance in various simulation scenarios. We compare our results to an ILP-provided optimal offline solution.

The paper is organized as follows: Sec. II overviews the related work. Sec. III describes our reference architecture and the the optimization problem in the form of an ILP to be solved. Our online heuristic algorithm is explained in Sec. IV. Performance measurements are evaluated in Sec. V. We conclude our work in Sec. VI.

II. Related Work

Virtual Network Embedding (VNE) is known to be NP-hard [7], which means finding the optimal solution cannot be done within reasonable time in case of large input, for example, when many services are deployed in a large infrastructure. Two different approaches exist to solve the VNE problem: i) exact solutions that find the optimum but these can be applied to limited scale problems, ii) approximation-based algorithms that trade the optimal solution for better runtime. [8] summarizes many of the possible solutions to the VNE problem.

Several approaches use Integer Linear Programming (ILP) to solve the VNE problem. In [9] the authors implemented an ILP formula to minimize the cost of embedding in terms of edge costs while maximizing the acceptance ratio. Reconfiguration of the existing mapping by enabling VNF migrations formed as MILP (Mixed ILP) were studied in [10].

Many different approaches solve the VNE problem with heuristic algorithms. Most of them perform the mapping in two steps: node mapping stage and edge mapping stage, thus physical nodes that have been selected to host neighboring network functions in the node mapping state may be multiple hops away from each other. Many algorithms aim to solve this problem by minimizing link utilization, e.g., [11], [12]. Authors of [13] proposed a hybrid algorithm, which first solves a relaxation of the original problem by using linear programming in polynomial time. Then they use deterministic and randomized rounding techniques on the solution of the linear program to approximate the values of the variables in the original MILP. A decomposing mapping algorithm proposed in [14] aims to minimize the mapping cost by making a selection of the available decompositions during the node mapping stage.

By Edge Computing, we mean a new network functionality, that extends the traditional cloud computing paradigm with additional computing capacity placed close to the end users. These resources are distributed in the service provider's network edge, for example co-located with an Internet Edge PoP (Point of Presence). This new approach makes possible to serve the users at the edge of the network rather than routing over the whole Internet backbone to the data centers located in the core, where all the computing capacity is concentrated. This ensures significant latency reduction and bandwidth savings on the backbone links, thus better QoS (Quality of Service) can be provided. The Open Edge Computing Initiative [15] is the responsible for driving the development of the Edge Computing technology.

The Open Fog Consortium [16] has proposed a possible reference architecture for the 5G ecosystem, called Fog Computing. The architecture [17] can be divided into three main layers. The top layer includes the central clouds that can be either the ISP's own private cloud or a public cloud provider's (e.g., Amazon Web Services, Microsoft Azure) infrastructure. The Fog Nodes, which can be found distributed in the ISP's network are located in the middle layer. They have less computing capacity compared to the previous layer, but can host applications with strict requirement on response time. While having limited resources, Fog Nodes can be used to enhance the performance of the end devices, or to offload the computation intensive tasks from them, thus ensuring better battery lifetime and response times. The bottom layer hosts the end-devices that consume the compute and network resources of the ISP. The devices are usually connected to the network via wireless interface. Further features of the end equipment are location-independence, limited hardware resources and large quantity.

Fog Computing also defines an ideal architecture for one of today's most important emerging paradigm, which is the IoT (Internet of Things) [18], [19]. In IoT, sensor devices in the bottom layer usually monitor different environmental variables, then send the measurement data to a central entity located in the network. This entity may send control messages back to the devices in order to change their state, then it aggregates and transmits the data to an other unit, that for example stores the data for later data processing and big-data applications. This other unit may be suitable to be placed in the central cloud infrastructure, because of the high storage requirements and the computational intensive data processing methods that can be performed more efficiently there. Other

Fog Computing related use-cases and challenges, for example from security point of view can be read in [20] and [21].

III. HETEROGENEOUS NETWORKS

By heterogeneous networks we mean an infrastructure, where different service providers are present. In this section, we introduce our network model based on Fog/Edge Computing, then we give the formal statement of the emerging resource allocation problem in heterogeneous networks.

A. Reference network

Based on the previously described considerations we created our own simplified network model for the three-tier Fog Computing architecture: a network consists of a given number of Fog Nodes (or Edge Computing Clusters) and Central Clouds. Each Fog Node contains a random number of servers with given computing capabilities (CPU, RAM and storage) and two gateway nodes. Each Fog Node has a SAP (Service Access Point) attached to it via the SAP-Gateway. The SAP represents the connection point to the network from the perspective of the end devices. They can reach the network resources through this interface (the SAP can be understood as for example a mobile base station). Within a Fog Node the servers are connected in a full mesh topology. We assume that bandwidth is not the bottleneck therein, because the nodes that belong to the same Fog Node are located close to each other and the blocking-less feature can be provided by choosing the right data center topology. The Fog Nodes and the central data centers are interconnected with each other via the Core Network. Each link has its custom delay and bandwidth characteristics assigned to it. The central cloud can be hosted by a public infrastructure provider (e.g., Amazon Web Services, Microsoft Azure) or the ISP's own data center. A topology may contain any number of central clouds. We assume that they have unlimited compute, storage and memory capacity, but the service provider may needs to pay a fee for the consumed resources. Fig. 2 shows an example topology with four fog clusters and one central cloud node.

B. Problem statement

We are searching for a solution to the following problem: How can we deploy service chains to a previously described heterogeneous cloud environment in a cost effective way? Let us assume that we are an ISP with Fog Nodes scattered around our network with given computing capabilities. Furthermore, we have access to one or more public cloud provider's infrastructure through the Internet. Because of economic reasons, it may be beneficial to have a contract with more than one provider, thus better prices can be achieved for the allocation. We expose our network to the end-users, who can then initiate SFC deployments with various QoS requirements. In this case, an efficient algorithm, which allocates physical resources to the components of the SFC, is necessary. The goal of the algorithm is to minimize the cost to be payed for consuming external resources by fulfilling the QoS requirements and

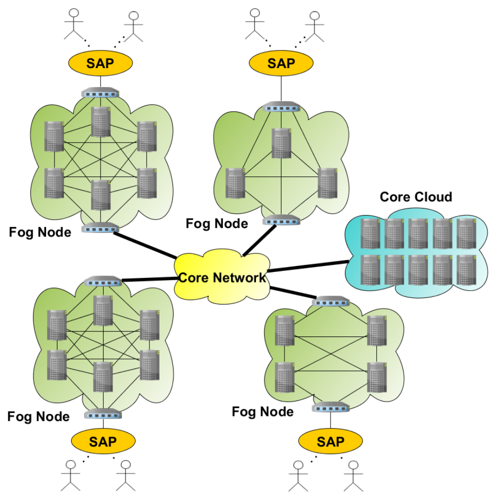

Figure 2. Example to the modeled architecture

other constraints that are dictated by the capabilities of the network. Finding an optimal solution is known to be NP-hard as this problem can be treated as a generalized version of the previously described VNE problem (arbitrary resource cost assigned to each node).

Table I
NOTATIONS USED

Notation	Description
V_s, E_s	NFs and links of the service graph
V_r, E_r	Nodes and links of the resource graph
$(i, j) \in E_s$	SG link between NF i and j
$(u, v) \in E_r$	RG link between node u and v
$x_u^i : i \in V_s, u \in V_r$	1 if NF i has been mapped to node u
$y_{u,v}^{i,j}$	1 if (u,v) is in the physical path of (i,j)
$\vec{r_i}$	Resources required by NF i
$\vec{\rho_j}$	Available resources in Node j
$\delta_{u,v}$	Delay of physical link (u,v)
$d^{i,j}$	Maximal delay between NF i and j
$\beta_{u,v}$	Available bandwidth on (u,v) link
$b^{i,j}$	Required bandwidth between NF i and j
$\varrho_s \subset V_s$	SAPs in the SG
$\varrho_r \subset V_r$	SAPs in the RG
$\varrho_s \subseteq \varrho_r$	SG sAPs can be found in the RG also
c_u^i	The cost of running NF i on Node u
$\gamma \subset V_r$	Core Cloud nodes

Equations 1-8 describe the problem as an ILP in the following section. The notations we use in the formal problem description are summarized in Table I. We provide the intuitive meaning of each line of the ILP in the following.

$$\forall i \in V_s : \sum_{u \in V_r} x_u^i = 1 \tag{1}$$

$$\forall (i,j) \in E_s, \forall u \in V_r : \\ \sum_{v:(u \to v) \in V_r} y_{u,v}^{i,j} - \sum_{w:(w \to u) \in V_r} y_{w,u}^{i,j} = x_u^i - x_u^j \tag{2}$$

$$\forall u \in V_r, \forall i \in V_s : \sum_{i \in V_s} x_u^i \vec{r_i} \leq \vec{\rho_u} \tag{3}$$

$$\forall (i,j) \in E_s : \sum_{(u,v) \in E_r} y_{u,v}^{i,j} \delta_{u,v} \leq d^{i,j} \tag{4}$$

$$\forall (u,v) \in E_r : \sum_{(i,j) \in E_s} y_{u,v}^{i,j} b^{i,j} \leq \beta_{u,v} \tag{5}$$

$$\forall i \in \varrho_s, : x_i^i = 1 \tag{6}$$

$$\min \sum_{u \in V_r} \sum_{i \in V_s} x_u^i c_u^i \tag{7}$$

As the result of the mapping, each VNF is assigned to exactly one physical node (1). The flow constraint is given by (2). The total amount of resources required by the VNFs mapped to a given node cannot exceed the resources available at the node (3). The total delay on the physical path of a SG link cannot exceed the requested delay between the two VNFs (4). Similarly, the total bandwidth of virtual links mapped to the same physical link cannot be greater than the available bandwidth (5). The SG SAPs are mapped to the RG SAPs with the same ID (6). The objective function of the optimization is to minimize the external resource costs (7). The cost of deploying a VNF is calculated based on the formula:

$$c_u^i = \alpha_u * CPU^i + \beta_u * RAM^i + \gamma_u * BW^i + \delta_u * STR^i, \tag{8}$$

where α, β, γ and δ are the cost parameters of the given physical node. The CPU, RAM, BW and STR are the resources requested by the VNF, where BW is the sum bandwidth on all links connected to the VNF, and STR is the allocated storage. In that case, when a node belongs to a fog nodes, we set the parameters to 0, thus ensuring that using our own infrastructure does not imply any cost. However, in real cloud environments currently Virtual machines usually created by using instance types with predefined CPU and memory resources and the additional bandwidth costs calculated based on the data volume moved to/from the cloud. In our model, we are using a more granular cost function. We entered the problem to an ILP solver by customizing the program that was used in [22] with our own cost calculation method.

IV. Our Online Algorithm

In this section we propose a novel orchestration algorithm that aims to minimize resource usage from the external core clouds, thus utilizing our own infrastructure in the most efficient way. VNF migration is an important feature of our approach, which gives option to migrate a given set of network functions to the cloud, thus freeing up network resources in the fog nodes in order to serve more latency and bandwidth sensitive requests. We introduce the $migrationcost$ attribute in (9):

$$migrationcost^i = \min \lambda_u * STR^i, u \in \gamma, \tag{9}$$

as there is a cost penalty to be payed for moving VNF_i to the cloud from one of the fogs. The rationale is that a migration process requires redundant resources to be allocated caused by the duplicating the state of the network function to be transfered to the cloud. We derive this cost from the migration coefficient of a node (λ) and the allocated storage to VNF^i. The main steps of our algorithm taking into account migration costs are described in the following, and pseudo-code is provided in Alg. 1 and 2.

A. SG preprocessing

In the first step of our algorithm, we calculate the order of execution. The ORDERSUBCHAINS method splits the incoming service request to the list of triplets containing the links and their connected nodes, i.e., the VNFs. The method starts with the first available SAP and collects all the neighboring nodes and connected edges, then appends the link with the strictest bandwidth requirement to the list together with its endpoints. After that it collects the available nodes and edges that became reachable via the new node.

B. VNF mapping

The next step step is the mapping of the service requests to the physical infrastructure. The MAP method iterates trough the previously ordered list of edges. Depending on the status of the nodes connected by the link, three different cases are possible. If both ends have already been allocated to a computing resource previously, then only a suitable path for the virtual edge needs to be found. To achieve this we run a Dijkstra algorithm between the hosts in the physical topology.

If one of the end nodes is not in mapped_vnodes yet and it is not a SAP either, then the node needs to be mapped. In the MAPVNF method the program tries to find a suitable place for the VNF. First, it filters the available physical nodes based on computing resources, and after that it checks if the candidate is reachable from the previous node via any sequence of edges. If the path does not satisfy the delay requirement, or any of the edges does not have enough free bandwidth, then the node is removed from the list of candidates. When the list of compatible physical nodes is available, they are sorted based on the resource cost of hosting the actual VNF. After the host node is determined, the link can also be mapped with the previously seen method. In that case, when the actual element is a SAP, then the algorithm calculates the path with the lowest latency, where the required bandwidth is available on all edges. If the path fulfills the latency requirement between the previously mapped VNF and the SAP, then the link mapping is performed.

It may occur, that one of the steps above fails. For example, none of the nodes have enough resource to host a given VNF, or the network related requirements cannot be met. In that case, the algorithm tries to step back to a previous state. This step is performed by the ROLLBACK method. In order to ensure better runtime, limiting the number of rollback steps may be necessary. We can do that by setting the max_rollback constant to an appropriate value. The ROLLBACK method

Algorithm 1 Service graph mapping to resource graph

```
1: $running \leftarrow$ copy($RG$)
2: $mapped\_vnodes \leftarrow \emptyset$
3: $map\_list \leftarrow$ORDERSUBCHAINS($SG$)
4: $mapped\_vnodes.insert(\varrho_s.first)$
5: $rollback\_level$ = 0
6: for all $(u, v, link) \in map\_list$ do
7:     if $(u, v) \in mapped\_vnodes$ then
8:         $success \leftarrow$ MAPVIRTUALLINK($link$)
9:     else if $u \notin \varrho_s$ then
10:        $success \leftarrow$ MAPVNF($u, v, link$)
11:    else                ▷ This means actual_element is a SAP
12:        $success \leftarrow$ MAPVLINK2SAP($u, v, link$)
13:    end if
14:    if $\neg success$ and $rb\_level \geq max\_rb$ then
15:        $success \leftarrow$ MIGRATINGEDGE2CORE($cable, u, v$)
16:    else
17:        $success \leftarrow$ ROLLBACK($u, v, link$)
18:        $rollback\_level += 1$
19:    end if
20:    if $success$ then
21:        $mapped\_vnodes.insert(v)$
22:    end if
23: end fordone
```

restores the state when the previous VNF was mapped, then chooses an other candidate from the list of the suitable nodes, and continues the mapping from the modified state. If the number of rollbacks exceeds the limit, then the algorithm tries to migrate one or more already mapped VNFs to the central cloud, thus freeing up resources in the fog nodes. The migration process is described in the next section.

C. Migrating VNFs to the cloud

The method that performs the migration can be divided into three parts. The first part collects the possible fog nodes containing VNFs that can be moved to the central cloud without violating the latency and bandwidth constraints. The GETFOGSFROMVNFS method returns the fog nodes which contain movable VNFs. If the fog node of the previously mapped VNFs is in the result, then that fog node is surely suitable to host the actual VNF in terms of network related constraints. In other cases, the GETPATH method collects the sequence of the physical links between the host of the previous VNF and the actual fog node. After that, if the performance of the physical path is conform with the virtual link requirements, then the fog node is stored as a possible element.

The second part of the method determines the migration options from the fog nodes selected previously. The DELUNCOMPVNF method removes the VNFs that belong to incompatible fog nodes from the list containing movable VNFs. After that, iterating through the list of the physical nodes that have movable VNFs on it, if by migrating the VNF from a node enough resources can be freed up, then we calculate the migration cost of the VNF and insert it to the migration

Algorithm 2 VNF migration from fog to cloud

```
1: procedure MIGRATINGEDGE2CORE($cable, u, v$)
2:     $compatible\_fogs \leftarrow \emptyset$
3:     $fog\_list \leftarrow$ GETFOGSFROMVNFS($cable$)
4:     for fog $\in$ fog_list do
5:         if $fog == u.fog$ then
6:             $compatible\_fogs$.insert($fog$)
7:         else
8:             $vlink \leftarrow$ GETVLINK($u, v$)
9:             $phy\_path \leftarrow$ GETPATH($u.fog$, $fog$)
10:            if CHECKPATH($phy\_path, vlink$) then
11:                $compatible\_fogs$.insert($fog$)
12:            end if
13:        end if
14:    end for
15:    DELUNCOMPVNFS($cable, compatible\_fogs$)
16:    $possible\_nodes \leftarrow$ GETPHYNODES($cable$)
17:    $mig\_list \leftarrow$ ordered empty list
18:    for node $\in$ possible_nodes do
19:        $vnf\_list \leftarrow$ ordered empty list
20:        for vnf $\in$ node.corable_vnfs do
21:            if ISMIGRABLE($vnf$) then
22:                $vnf.mig\_cost \leftarrow$ GETMIGCOST($vnf$)
23:                $mig\_list$.add($vnf$)
24:            end if
25:        end for
26:        EXPANDMIGLIST($mig\_list, node.cable\_vnfs$)
27:    end for
28:    for mig_opt $\in$ mig_list do
29:        DOMIGRATING($mig\_opt$)
30:        if ISMIGWASSUC($mig\_opt$) then
31:            return True
32:        else
33:            RESTOREMIG($mig\_opt$)
34:        end if
35:    end for
36:    return False
37: end procedure
```

list. The migration list is ordered by the migration cost of the contained elements. It may occur that by moving only one VNF to the cloud at a time the required amount of resources cannot be provided to the new NF. Because of that, checking different subset of VNFs mapped to the same physical node may be necessary. Instead of checking all the possible subsets, in order to reduce runtime we only test the neighboring pairs, triplets, and so on in the EXPANDMIGLIST method. If any of these subsets grants enough free resources, then we insert it to the migration list with the total migration cost of all the affected VNFs. The result of the process is an ordered list that contains the possible migrating options.

In the last step, the DOMIGRATING method tries to execute the migrating options from the migration list, starting with the cheapest option. The ISMIGRATING method checks in each iteration if the migration was successful. If the remapped physical paths fulfill the corresponding virtual link requirements, then the method returns true. Else in RESTOREMIG we restore the previous state and continue with the next migration option from the list.

V. MEASUREMENT RESULTS

We have run measurements to demonstrate how our algorithm works in different topology setups with various requests. We had six scenarios each of them contained different number of fog nodes between 1 and 20. During our simulations we posted the requests (the service graphs) one by one, till the first failure occurred. We indicated failure when there was no way to deploy a service graph completely to the remaining available resource set.

First of all, we compared our algorithm's efficiency with and without the migration function. In the comparison we examined how many CPUs our algorithm can deploy to the same resource set. It is easy to see that migrating is mostly used for cost efficiency, because if we turn off this feature then our algorithm has two options for deploying VNFs. First option: our algorithm deploys cloud compatible VNFs directly in the cloud. This is obviously not cost efficient because we have to pay for allocated resources to the third-party from the beginning. Second option: we want our algorithm to be cost efficient, so it puts all the VNFs in the fog nodes. It is definitely cheap for us, however our resources in the fog nodes will quickly get exhausted.

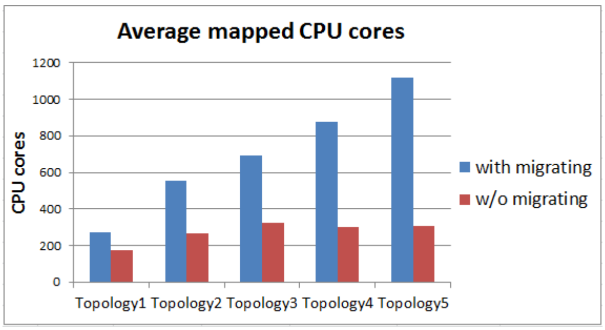

Figure 3. CPUs mapped with and without migration

Fig. 3 shows how many virtual CPU our algorithm can deploy with migrating and without migrating them if we want to keep our cost efficiency. Each bar on the figure represents the average number of mapped virtual CPUs on the quoted topologies. As you can see, we deploy more virtual CPUs when the migration feature is turned on.

Next, we wanted to examine what the cost difference between our algorithm and an optimal solution is. The optimal solution comes from the offline algorithm, which solves the previously defined ILP problem. Both algorithms were used with the same input, which contained the same resource graph and the same set of request graphs. We executed our algorithm and collected all the successfully deployed requests till the first failure. After that, we executed the offline algorithm with the same resource graph and the collected request set. Both

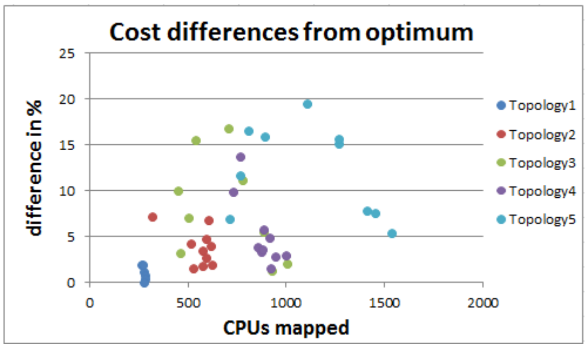

Figure 4. Differences of cost between the online and offline algorithms

algorithms return with a number that defines the price of the request set on the given topology. We compared these outputs and plotted those in a scatter plot shown in Fig. 4: each point represents a cost difference for a given request sequence. It is noticeable how the request randomization affects the deployed CPU number. It is also remarkable how our algorithm scales: with the increase of the number of fog nodes the difference between the two solutions does not go higher than 20%.

Table II
RELATIVE STANDARD DEVIATION OF COSTS

	Mean of cost differences	Relative Standard Dev.
Topology1	0,9841	0,7931
Topology2	3,8272	0,5235
Topology3	8,0787	0,7032
Topology4	5,2071	0,7231
Topology5	12,1877	0,4061

Table II shows the calculated relative standard deviation based on the results displayed in Fig. 4. It tells us how the different results in each measurement are scattered around the mean cost. We can observe that as we increase the number of fog nodes significantly, the relative standard deviation starts to decrease, which also a proof of the good scalability.

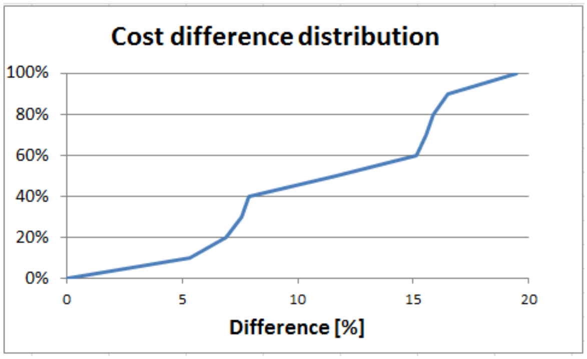

Figure 5. Cost difference distribution

In Fig. 5 we depict the distribution of cost differences. The x axis represents the cost difference (in percentage) between the offline and our algorithm. The optimal solution contains

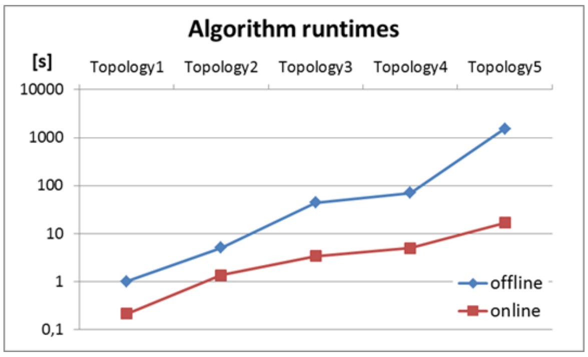

Figure 6. Runtimes of the algorithms

only the price of the allocated resource in the third-party environment (cloud). In our solution we calculate a migration cost as well. It is remarkable how the migration cost affects our final price. If the migration cost is higher than the resource allocation cost, then it is worthy to put a cloud compatible VNF directly to the core cloud, so we can avoid high migration cost.

Finally, we compared the execution times for both algorithms. The result is represented in Fig. 6. The offline algorithm's execution time is increasing exponentially with the number of fog nodes, however our algorithm execution time grows only linearly.

VI. CONCLUSION

In this work we examined how future networks should handle IoT services and their deployment. We made network topology examples that could be used to run such services. For handling the service creation on these topologies we created an online algorithm that can deploy services cost efficiently and that is also able to handle networking requirements of the services. Our algorithm runs in polynomial time, thus scales as well, while it provides close-to-optimal orchestration results.

REFERENCES

[1] N. Panwar *et al.*, "A Survey on 5G," *Phys. Commun.*, vol. 18, no. P2, pp. 64–84, Mar. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.phycom.2015.10.006

[2] ETSI, "White Paper: Network Functions Virtualisation (NFV)," 2013. [Online]. Available: http://portal.etsi.org/nfv/nfv_white_paper2.pdf

[3] ETSI GS NFV-PER 001, Dec 2014, *Network Functions Virtualisation (NFV); Architectural Framework*, ETSI, 2014 Dec.

[4] G. Brown, "Mobil edge computing use cases and deployment options," Tech. Rep., 2016.03.01. [Online]. Available: https://www.juniper.net/assets/us/en/local/pdf/whitepapers/2000642-en.pdf

[5] P. Mach *et al.*, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[6] J. Halpern *et al.*, "Service Function Chaining (SFC) Architecture," IETF RFC 7665, Oct. 2015.

[7] E. Amaldi *et al.*, "On the computational complexity of the virtual network embedding problem," *Electronic Notes in Discrete Mathematics*, vol. 52, pp. 213 – 220, 2016, INOC 2015 – 7th International Network Optimization Conference.

[8] A. Fischer *et al.*, "Virtual network embedding: A survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[9] I. Houidi *et al.*, "Virtual network provisioning across multiple substrate networks," *Computer Networks*, vol. 55, no. 4, pp. 1011–1023, 2011.

[10] G. Schaffrath *et al.*, "Optimizing long-lived cloudnets with migrations," in *UCC '12, ACM*. ACM, 2012, pp. 99–106.

[11] H. Cui *et al.*, "A virtual network embedding algorithm based on virtual topology connection feature," *Wireless Personal Multimedia Communications*, 2013.

[12] G. Wang *et al.*, "A virtual network embedding algorithm based on mapping tree," *Communications and Information Technologies*, 2013.

[13] M. Chowdhury *et al.*, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 206–219, 2012.

[14] S. Sahhaf *et al.*, "Network service chaining with optimized network function embedding supporting service decompositions," *Computer Networks*, vol. 93, no. 3, pp. 492–505, 2015.

[15] Open Edge Computing Initiative. [Online]. Available: http://openedgecomputing.org

[16] OpenFog Consortium, "Openfog reference architecture for fog computing," Tech. Rep., 2017.02.08. [Online]. Available: https://www.openfogconsortium.org/wp-content/uploads/OpenFog_Reference_Architecture_2_09_17-FINAL.pdf

[17] C. C. Byers, "Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for fog-enabled iot networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 14–20, 2017.

[18] A. V. Dastjerdi *et al.*, "Fog computing: Helping the internet of things realize its potential," *IEEE Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[19] F. Bonomi *et al.*, "Fog computing and its role in the internet of things," *MCC '12*, pp. 13–16, 2012.

[20] S. Yi *et al.*, "A survey of fog computing: Concepts, applications and issues," in *Mobildata '15, ACM*. ACM, 2015, pp. 37–42.

[21] L. M. Vaquero *et al.*, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[22] B. Németh *et al.*, "Efficient Service Graph Embedding: A Practical Approach," in *Second IEEE International Workshop on Orchestration for Software Defined Infrastructures (O4SDI @ IEEE NFV-SDN 2016)*, Nov 2016.

Marton Szabo received his M.Sc. degree in informatics from Budapest University of Technology and Economics (BME) in 2017. During his studies he was a member of the High Speed Networks Laboratory hosted by the Department of Telecommunications and Media Informatics. After working in the telecom industry for one year he joined the newly formed MTA-BME Network Softwarization research group founded by the Hungarian Academy of Sciences (MTA). His current research focuses on network softwarization and virtualization in 5G.

Mark Szalay is a graduate student at Budapest University of Technology and Economics. He is a member of the High Speed Network Laboratory (http://hsnlab.tmit.bme.hu/) in the Department of Telecommunications and Media Informatics. His main research interests include Hardware (Router/switch/NIC) design, Network programming, Software-Defined Networking (SDN) and Network Function Virtualization (NFV). Mark has been working in this field for more than two years.

David Haja is an undergraduate student at Budapest University of Technology and Economics. He is a member of the High Speed Network Laboratory (http://hsnlab.tmit.bme.hu/) in the Department of Telecommunications and Media Informatics. His main research interests include Software-Defined Networking (SDN), Network Function Virtualization (NFV) and Resource Orchestrartion.