Journal of Network and Computer Applications 170 (2020) 102785

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

# Scalable edge cloud platforms for IoT services

Balázs Sonkoly[a,b,*], Dávid Haja[a,b], Balázs Németh[a,b], Márk Szalay[a,b], János Czentye[a,b], Róbert Szabó[c], Rehmat Ullah[e], Byung-Seo Kim[d], László Toka[a,b]

[a] *MTA-BME Network Softwarization Research Group, Hungary*
[b] *Budapest University of Technology and Economics, Hungary*
[c] *Ericsson Research, Hungary*
[d] *Hongik University, Sejong, South Korea*
[e] *Gachon University, Seongnam, South Korea*

ARTICLE INFO

*Keywords:*
Edge computing
Resource orchestration
SDN
NFV
IoT

ABSTRACT

Nowadays, online applications are moving to the cloud, and for delay-sensitive ones, the cloud is being extended with edge/fog domains. Emerging cloud platforms that tightly integrate compute and network resources enable novel services, such as versatile IoT (Internet of Things), augmented reality or Tactile Internet applications. Virtual infrastructure managers (VIMs), network controllers and upper-level orchestrators are in charge of managing these distributed resources. A key and challenging task of these orchestrators is to find the proper placement for software components of the services. As the basic variant of the related theoretical problem (Virtual Network Embedding) is known to be $\mathcal{NP}$-hard, heuristic solutions and approximations can be addressed. In this paper, we propose two architecture options together with proof-of-concept prototypes and corresponding embedding algorithms, which enable the provisioning of delay-sensitive IoT applications. On the one hand, we extend the VIM itself with network-awareness, typically not available in today's VIMs. On the other hand, we propose a multi-layer orchestration system where an orchestrator is added on top of VIMs and network controllers to integrate different resource domains. We argue that the large-scale performance and feasibility of the proposals can only be evaluated with complete prototypes, including all relevant components. Therefore, we implemented fully-fledged solutions and conducted large-scale experiments to reveal the scalability characteristics of both approaches. We found that our VIM extension can be a valid option for single-provider setups encompassing even 100 edge domains (Points of Presence equipped with multiple servers) and serving a few hundreds of customers. Whereas, our multi-layer orchestration system showed better scaling characteristics in a wider range of scenarios at the cost of a more complex control plane including additional entities and novel APIs (Application Programming Interfaces).

## 1. Introduction

Fog and edge computing are novel concepts extending traditional cloud computing approach by deploying compute resources closer to customers and end devices. Although the two concepts are similar since both shift the computation and storage closer to the edge of the network, they are not identical. As the authors of Yousefpour et al. (2019) and Ren et al. (2019) emphasize, fog computing has an n-tier hierarchical architecture that means all the network devices along the routing path between the end device and the cloud can provide computing, networking, storage, control, and acceleration services. In contrast, edge computing tends to be limited to computing at servers deployed one (or a few) hop away from the end devices typically at macro or micro base stations. Apart from the differences, both approaches enable several future 5G applications and network services, such as IoT applications, Tactile Internet, AR/VR (augmented/virtual reality) use-cases, or remote driving. Edge resources provide execution environments close to users in terms of latency (e.g., in mobile base stations). By these means, on the one hand, customers' devices can offload computational tasks to this environment instead of consuming their local resources. On the other hand, latency-critical functions can be offloaded from central clouds to the edge enabling critical machine type communication which is required by various envisioned services.

A dedicated component, namely the resource orchestrator (RO), is in

* Corresponding author. Budapest University of Technology and Economics, Hungary.
*E-mail address:* balazs.sonkoly@tmit.bme.hu (B. Sonkoly).

https://doi.org/10.1016/j.jnca.2020.102785
Received 24 January 2020; Received in revised form 3 July 2020; Accepted 27 July 2020
Available online 29 August 2020

charge of finding the proper placement of software components realizing the service. Following ETSI's[1] terminologies on Network Function Virtualization (NFV) (White Paper, 2013), the software modules composing the network service are referred to as Virtual Network Functions (VNFs). RO can be considered as a component encompassing orchestration related tasks, and in ETSI's architecture it appears both in the Virtual Infrastructure Manager (VIM) and in the NFV Orchestrator (NFVO). In general, RO assigns VNFs composing the service to compute resources and also allocates paths between connected VNFs.

A novel RO (or a hierarchy of ROs) which can efficiently manage underlying resources in edge cloud or mobile edge computing environments is an inevitable future component with challenging tasks. It must be able to jointly handle compute and network resources in a tightly integrated framework and it must be aware of network characteristics besides computing capabilities. Furthermore, the requested network services have to be created on-the-fly within seconds. It is challenging as even the problem of Virtual Network Embedding (VNE) is known to be $\mathcal{NP}$-hard (Rost and Schmid, 2020) which addresses only the mapping of network elements. Two different design approaches can be applied to achieve such features. On the one hand, the VIM itself can be extended with network-awareness and with the detailed view on network resources. This feature is typically not available in today's VIMs. With such an upgrade, the additional NFVO becomes unnecessary for single-provider setups where resources belong to the same operator, and by these means, the orchestration and deployment time can be reduced significantly. Apparently, this approach can be feasible only for smaller systems encompassing a limited number of resource pools due to scalability issues. On the other hand, on top of VIMs and network controllers, a higher level orchestrator, i.e., the NFVO, can be added which is able to combine/integrate different resource domains. This solution results in a hierarchy of ROs and the cooperation of VIMs and NFVO yielding larger deployment time and the need for strictly defined external APIs. However, multi-provider scenarios require this approach (Gerő et al., 2017; Vaishnavi et al., 2018), and it yields a scalable solution for larger networks.

In this paper, we propose fully-fledged solutions for both approaches and evaluate their performance characteristics. As today's most widely deployed open-source VIM is OpenStack, we target that platform. As a first solution, we propose and implement a novel extension to OpenStack which makes it a network-aware resource orchestrator. As a result, a single OpenStack system will be in charge of controlling both the cloud and edge resources and the VNFs are implemented as virtual machines (VMs). For more complex multi-provider scenarios, we propose a multi-layer orchestration system where cloud resources are managed by OpenStack while edge resources are under the control of Docker. This requires Docker engines to be installed on edge servers. We assume that at the edge of the system limited amount of compute resources are available, therefore to reduce the virtualization overhead in our multi-provider scenarios we opt for using light-weight software containers instead of virtual machines. Both VIMs (OpenStack and Docker) are extended with a common resource control API and a multi-domain NFVO is added on top. We assume that each VNF can be run as a VM or as a container and the NFVO selects the preferred deployment option on-the-fly.

Our contribution is threefold. First, we define the two architecture proposals, highlight the main benefits and the limitations. We also provide online embedding algorithms together with information models adjusted to the architectures, respectively. These algorithms are the key components of the orchestration systems significantly affecting the overall performance. Second, we implement proof-of-concept prototypes capturing the relevant parts of the proposed systems. And finally, we evaluate the concepts via large-scale simulations and real experiments to reveal the scalability characteristics of both approaches. In order to enable realistic experiments with both prototypes, we set up two dedicated, fully operational testbed environments including multiple blade servers and the whole software stacks. The results confirm that the simple VIM extension can be a valid option for single-provider setups up to 100 edge Points of Presence (PoPs) and serving a few hundreds of customers, whereas our multi-layer orchestration system shows better scaling characteristics in terms of the number of clients and in the network size. More exactly, our mapping algorithm supports thousands of users and hundreds of edge PoPs, while its operation overhead is minimal, compared to a *de facto* standard system which cannot handle latency constraints.

The rest of the paper is organized as follows. In Sec. 2, we highlight an envisioned service as an illustration. Sec. 3 is devoted to the architecture proposals and the corresponding resource orchestration algorithms. Sec. 4 describes our proof-of-concept prototypes. In Sec. 5, we present our main results with the two implemented prototypes and reveal the main performance characteristics. In Sec. 6, a summary of the related work is given while Sec. 7 draws the conclusion.

## 2. An illustrative use-case

We chose an automotive use case in order to illustrate the power of the edge cloud computing concept: an alerting system that raises drivers' attention to road dangers ahead. Although this type of service already exists,[2] its implementation as a novel IoT application in a distributed system offers improved capabilities to the existing ones'. The advancement stems from the round-trip delay saving, i.e., faster alerts to drivers, provided by the local edge instead of the remote central cloud computing infrastructure.

The schematic description of the envisioned IoT application is depicted in Fig. 1. The service is composed of multiple components. The Object recognition (OREC) function instances process images and video streams uploaded from cars that are being driven by customers/clients of the application provider, connected to wireless access networks, e.g., 4G mobile service. The OREC instances are deployed at the edge of the network as close to the cars as possible in order to ensure the lowest latency until object recognition. Hazardous objects that these functions look for in the input data can be wild animals crossing the road, storm destructed trees laying across the asphalt, or biker in the city, kids around the school, etc. OREC instances process all upload streams and feed instant alerts back to the respective drives if necessary. Prediction engines (PRED) are deployed possibly co-hosted with OREC instances in

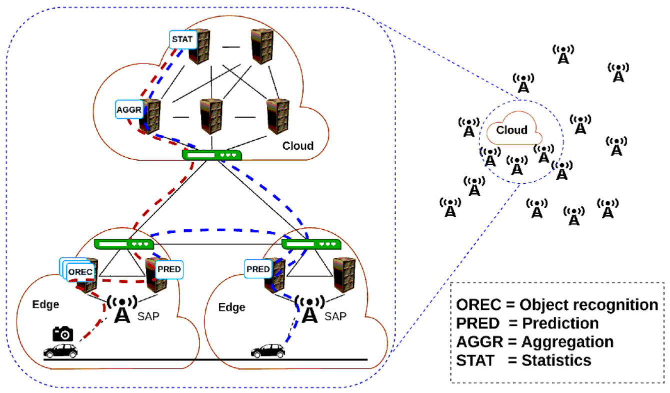


**Fig. 1.** The actors and components of the distributed road danger alerting system.

[1] European Telecommunications Standards Institute (ETSI), https://www.etsi.org/.

[2] https://www.here.com/products/automotive/hazard-warnings.

order to forecast imminent and short-term dangers based on the recognized hazardous situation, e.g., animals approaching the road. Both components can emit alerts to the drivers in case their own car or the cars ahead uploaded visual data that were flagged as an instant or imminent danger situation. The alert messages are displayed to the driver or even trigger automatic emergency braking if needed. Alerts can also be sent to those customers who do not provide video uploads to the system due to lack of integrated or dash cams.

Similarly to centralized existing services, the system comprises aggregating (AGGR) and statistical (STAT) function components in data centers. As their delay requirements are not considered critical, these functions are better not to be placed on scarce edge resources. The AGGR instances correlate and aggregate the events reported by the OREC and PRED functions scattered in the edge infrastructure. The function can be scaled according to the actual demand. For overall statistics and the continuous learning of the recognition and prediction features, the centralized STAT components collect all important data from the system and send the training data back to OREC and PRED functions.

Note that all types of components can be dynamically stopped, started, migrated across the whole infrastructure as needed. Migration of functions might be required for stateful PRED components as those must follow drivers on their tracks. In Fig. 1 we depict with dashed bubbles the edge domains (EDGE) and the data center (CLOUD). Furthermore, by exploiting the possibilities of the edge computing concept, the edge-deployed PREDs can provide inputs directly among themselves, e.g., predicting hazardous situations for cars. Therefore the inputs can be handled and being processed by other edge domains, following the car based on whose video feed a danger had been detected.

## 3. Proposed architectures and algorithms

In order to enable future services with strict latency bounds, such as the envisioned example shown in Sec. 2, geographically distributed resources have to be managed and controlled carefully in an integrated system. Resource orchestration in distributed cloud environments is a challenging task and novel algorithms, workflows and architectures are needed to meet application level requirements. Here, we present two architecture proposals with different capabilities together with the key algorithms responsible for resource orchestration.

### 3.1. DARK: extended VIM for a single provider

The first and simpler architecture option is to extend a VIM in order to support edge cloud infrastructures. Today's VIMs typically designed for data center environments where the resources (compute, storage, network) are placed close to each other in a central premises and the well-designed network topology provides extreme bisection bandwidth. To put it simply, we have zero delay and infinite throughput between the VMs. Thus, widely used scheduler algorithms do not take the network characteristics into consideration. In this section, we propose a novel orchestration algorithm, called DARK, that aims to cope with the new challenges of distributed cloud architectures where delays cannot be ignored. It provides a general extension to traditional VIMs by adding "network-awareness" to the resource orchestration process. The basic version of the algorithm was described in Haja et al. (2018). Here, we introduce the resource model including network topologies, the service model and summarize the main steps of our heuristics.

#### 3.1.1. Resource and service models

Our network model for a three-tier edge computing architecture consists of a given number of edge clusters and central clouds connected via the core network as it is shown in the example in Fig. 2. In DARK, the physical architecture is represented as a graph called resource graph (RG). Each edge cluster contains a pre-defined number of servers with given computing capabilities (CPU, RAM and storage) and two gateway nodes. Each of the clusters has a SAP (Service Access Point) attached to it

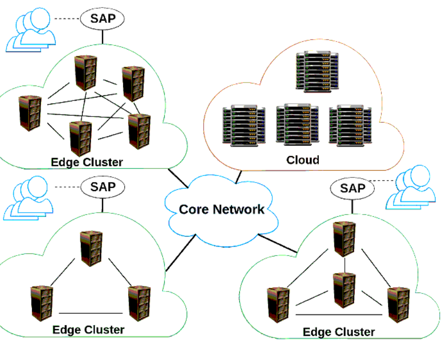


**Fig. 2.** DARK's resource model.

via the SAP-Gateway. The SAP works as a connection point to the network. The end devices can consume the remote resources through this interface (e.g., a mobile base station). We assume that edge clusters, including their servers, have preconfigured, in typical scenarios, limited, computational resources (from one single server to a few number of rack cabinets) that we consider during the deployment of services. Within a cluster, the nodes are connected in a full mesh topology. The edge clusters and the cloud data centers are connected to each other via the core network. Each link in our graph is weighted with the latency and bandwidth value corresponding to the physical connection characteristics. A topology may contain any number of cloud domains that are owned either by the service provider or another operator (e.g., Amazon). We assume that cloud data centers have much more compute, storage and memory capacity, than the edge clusters. In cases, when other operator's resources are considered the service provider has to pay a fee for consumed resources according to a cost model. However, in this architecture option, we assume a single provider using only her own resources.

Our novel orchestration algorithm takes a set of Service Function Chains (SFC) as its input. More complex services are generally described by abstract Service Graphs (SG) encompassing multiple chains. This representation can easily be transformed into a set of SFCs. An example transformation and our processed inputs are shown in Fig. 3 as an illustration. Each SFC contains a SAP as its starting point and numerous Virtual Network Functions (VNFs) realize the computational tasks in the service. In Fig. 3, each color represents different instances of a given VNF type. We consider the edges between the virtual nodes as bidirectional virtual links, which may have bandwidth and delay requirements. The delay requirements define the maximum tolerated latency between the two nodes and the given bandwidth requirement specifies the minimum throughput for the virtual link.

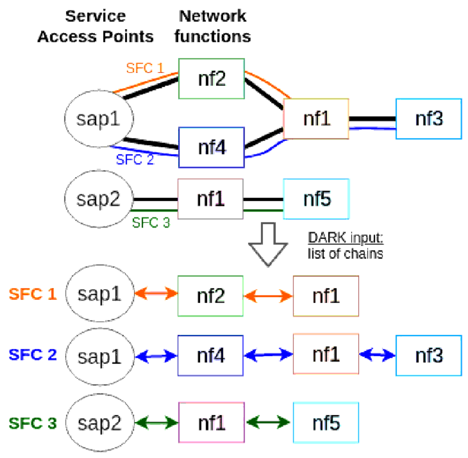


**Fig. 3.** DARK's service model (set of chains).

### 3.1.2. *Mapping algorithm*

The key component of DARK is our online mapping algorithm which maps the incoming service requests, given as service graphs, to the resource graph, describing the current state of the infrastructure and the already deployed services. Our greedy, heuristic solution processes SFCs starting from a SAP, moves forward on the virtual links toward connected VNFs and deploys the virtual entities on the physical system step-by-step. The pseudo-code of our algorithm is provided in Alg. 1. An important feature of our approach is the ability of VNF migration: moving a given set of already deployed network functions to the cloud or to other edge clusters if necessary. By these means, resources in given edge domains can be released in order to admit more services with strict latency constraints.

**Algorithm 1**
Service graph mapping to resource graph.

```
procedure MAP($SG$, $RG$, $migratable\_vnfs$)
  $running \leftarrow$ copy($RG$)
  $mapped\_vnodes \leftarrow \emptyset$
  $map\_list \leftarrow$ ORDERSUBCHAINS($SG$)
  $mapped\_vnodes.insert(SG.saps.first)$
  $rollback\_level = 0$
  for all ($u$, $v$, $link$) $\in map\_list$ do
    if $u \in mapped\_vnodes$ and $v \in mapped\_vnodes$ then
      $success \leftarrow$ MAPVIRTUALLINK($link$)
    else if $u \notin SG.saps$ then
      $success \leftarrow$ MAPVNF($u$, $v$, $link$)
    else                    ▷ This means actual_element is a SAP
      $success \leftarrow$ MAPVLINK2SAP($u$, $v$, $link$)
    end if
    if $\neg success$ and $rb\_level \geq max\_rb$ then
      $success \leftarrow$ MIGRATING($migratable\_vnfs$, $u$, $v$)
    else
      $success \leftarrow$ ROLLBACK($u$, $v$, $link$)
      $rollback\_level += 1$
    end if
    if $success$ then
      $mapped\_vnodes.insert(v)$
    end if
  end fordone
end procedure
```

The *MAP* function receives three parameters: the actual service request represented as a service graph (SG), the physical architecture as a resource graph (RG) and a list called *migratable_vnfs* that contains previously deployed VNFs, which might be migrated to other physical hosts. After initialization, in the first step of our algorithm, we determine the order of execution, which can have a significant impact on the final result of the greedy algorithm. Since our mapping process begins at a SAP and moves forward on the virtual links, a key requirement of mapping VNFs is that we need to be able to determine the reference nodes in the system, from where we have to fulfill the defined network requirements in the SFC. According to this, we have to split the incoming service request into a sequence of subchains, where each subchain holds the following elements: *i*) a VNF that is already processed or a SAP; *ii*) another virtual node to be examined; *iii*) a virtual link that connects the previous two nodes and defines the network requirements between them. According to our intuition, the deployment of VNFs connected with virtual links defining tighter latency bounds is more complicated and we have fewer options for hosting them. Therefore, we want to sort the previously described subchains based on the strictness of their delay requirements, but keeping in mind that at least one node in the subchain must have been mapped before, when our mapping algorithm reaches that subchain during its execution. The ORDERSUBCHAINS method is responsible for this ordering. More specifically, it splits the incoming service request into a list of triplets (subchain) containing the *links* and their connected nodes *(u,v)*, i.e., the VNFs.

The next step is the mapping of the service request to the underlying physical infrastructure. The MAP method iterates trough the previously ordered list of triplets. During the mapping we refer to the elements of each triplet as: previous element, actual element and virtual link. The steps of processing a triplet are presented by the flowchart shown in Fig. 4. Depending on the status of the nodes connected by the virtual link, three different cases are possible.

*Mapping virtual link between VNFs*: 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. The MAPVIRTUALLINK method will find this path if it exists. If both of the virtual elements are in the same cluster, we are done, since we assume that there is no network bottleneck inside a cluster. Anyway, with Dijkstra's algorithm we determine the shortest path, in terms of network latency, between the hosts in the physical topology. If the found path's latency is lower than the required, and all the links of the path have enough available bandwidth for hosting the new virtual link, then we can map the requested virtual link in the topology. This method can be reviewed on the right side of Fig. 4.

*Mapping VNF*: If the actual element is a VNF and it is not deployed yet, the MAPVNF function will map it to the underlying system. The core steps of MAPVNF is presented in the middle branch of the flowchart. First, it filters the available physical nodes based on their 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 latency requirement, or any of the edges do not have enough bandwidth, then that physical node is removed from the list of candidates. When the list of compatible physical nodes is available, they will be sorted based on the resource cost of hosting the actual VNF. After the host node is determined, the link can also be mapped by applying the previously seen method (MAPVIRTUALLINK).

*Mapping virtual link between VNF and SAP*: In that case, when the actual element is a SAP, then the algorithm calculates the path – starting from the previously mapped VNF (previous element) – with the lowest latency, where the required bandwidth is available on all edges. If the path fulfills the latency requirement, the virtual links are mapped to the

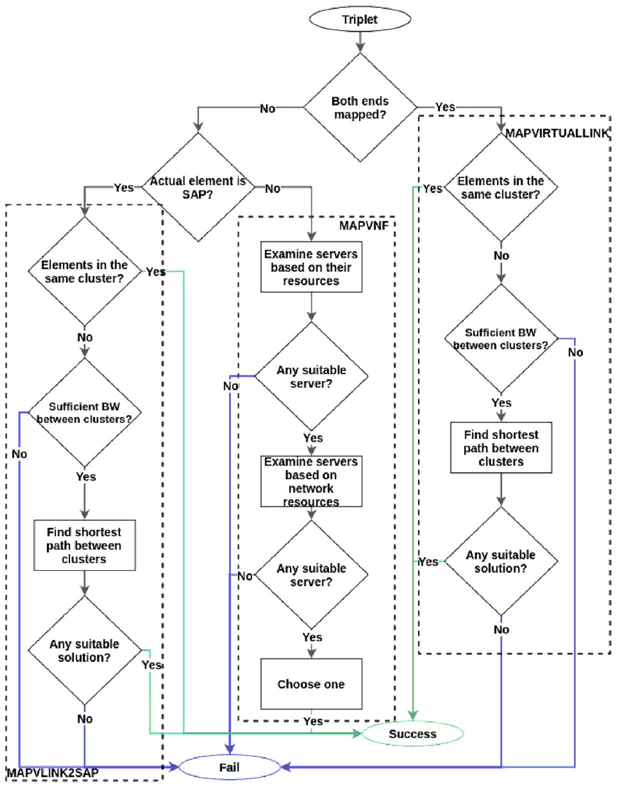

Triplet
Both ends mapped?
No
Yes
Actual element is SAP?
Yes
No
MAPVIRTUALLINK
Elements in the same cluster?
Yes
No
Sufficient BW between clusters?
No
Yes
Find shortest path between clusters
Any suitable solution?
Yes
No
MAPVNF
Examine servers based on their resources
Any suitable server?
No
Yes
Examine servers based on network resources
Any suitable server?
No
Yes
Choose one
Yes
Elements in the same cluster?
Yes
No
Sufficient BW between clusters?
No
Yes
Find shortest path between clusters
Any suitable solution?
Yes
No
Success
Fail
MAPVLINK2SAP

**Fig. 4.** Flowchart of DARK's mapping algorithm.

corresponding physical links. This MAPVLINK2SAP method works similarly as MAPVIRTUALLINK and presented on the left side of Fig. 4.

*Rollbacking*: It may occur that during the mapping of a given service chain, one of the above steps fails, which is noted with the “Fail” circle in the flowchart of Fig. 4. For example, none of the nodes have enough resources 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. By setting the *max_rollback* constant to an appropriate value, the service provider can limit the number of rollback steps in order to ensure an acceptable runtime. In each step of the ROLLBACK method, we restore a state that was previously achieved, modify that state, then continue the mapping from that modified status. In practice, let us consider that we cannot deploy the actual element due to some reasons. We call the ROLLBACK method that examines the suitable nodes for the previous element and chooses a different one than the MAPVNF chose previously. We consider this state modification as one rollback step. If we still cannot map the actual element, then we choose another physical node from the suitable nodes of the previous element. In that case, when we tried all the suitable nodes for the previous element, the ROLLBACK takes another step back in the SFC. By these means, we try to relocate each previous VNF in the SG beginning from the previous element – in the current triplet – to the very first VNF of the service, until we can successfully deploy the actual VNF or the amount of ROLLBACK steps reaches the *max_rollback* limit. If the number of rollbacks exceeds the *max_rollback* limit, then the algorithm tries to migrate one or more already mapped VNFs to another cluster or even to the cloud, thus freeing up resources in the edge cluster. It is worth noting that our algorithm does not deploy service components separately, thus either all components of a service request are successfully deployed or the request is rejected.

### 3.1.3. Migrating VNFs

A central and novel feature of the DARK algorithm is the support of VNF migration. This process is essential to enable dynamic operations and adaptability to varying workload or changing environment (e.g., moving car). According to our resource model, two migration directions may occur: *i*) from edge server to the cloud and *ii*) from edge server to edge server. On the one hand, the goal of the former is usually cost optimization. On the other hand, the latter is essential for fast adaptation, but it also increases the chance of successful deployment of a VNF into the edge server. In our algorithm, MIGRATING method attempts to migrate a previously deployed VNF from the given server in order to free up enough computation resources for the newly arrived VNF to deploy. Its workflow can be described in three phases: *i*) detecting the list of possible VNFs to migrate, and collecting where they could be relocated, *ii*) on the temporary resource model executing the migration and *iii*) checking whether the migration violates the network requirements of the migrated VNF.

Being central to our concept, we discuss our MIGRATING method in more detail and it is described as a pseudo-code in Alg. 2.

```
 1: procedure MIGRATING($migratable\_vnfs, u, v$)
 2:     $mig\_vnf\_try = 0$
 3:     for migratable_vnf ∈ migratable_vnfs do
 4:         if ISLARGER($migratable\_vnf$, $u$) then
 5:             if $mig\_vnf\_try < max\_vnf$ then
 6:                 $poss\_nodes$ ← GETCOMPNODES($migratable\_vnf$)
 7:                 $mig\_try = 0$
 8:                 for n ∈ poss_nodes do
 9:                     if $mig\_try < max\_try$ then
10:                         $backup\_rg$ ← GETRUNNINGRG()
11:                         $backup\_sgs$ ← GETMAPPEDSGS()
12:                         $success$ ← TRYMIGRATE($n$,$migratable\_vnf$,$u$,$v$)
13:                         if $success$ then
14:                             return True
15:                         else
16:                             RESETRG($backup\_rg$)
17:                             RESETSGS($backup\_sgs$)
18:                         end if
19:                         $mig\_try += 1$
20:                     end if
21:                 end for
22:                 $mig\_vnf\_try += 1$
23:             else
24:                 return False
25:             end if
26:         end if
27:     end for
28:     return False
29: end procedure
```

It returns *true* or *false* depending on the migration was successful or not. The three input arguments consist of the (*migratable_vnfs*) previously mentioned list in Alg. 1 of non-delay-sensitive network functions, (*u*) the actual VNF to be deployed and (*v*) the previous VNF in the service function chain connected to *u*. Please note that the VNF *v* is already handled by the embedding algorithm, so it is deployed to a physical node of the computing system.

The migration procedure could be summarized in the following three phases. First (lines 2–7), it iterates through the list containing migratable functions (i.e., non-delay-sensitive VNFs) and checks the compute constraints using ISLARGER method in lines 3 and 4. The ISLARGER method returns *true* if the consumed compute resources (CPU, RAM, and storage) of the migratable VNF are larger than the required by *u*. In this case, VNF *u* could be placed into the physical node instead of the migratable VNF. In the line 6, the GETCOMPNODES method returns the list of possible nodes where the migratable VNF could be migrated to.

The second phase of the MIGRATING method (lines 8–12) iterates through these possible physical nodes and attempts to execute the migration process. If the method did not reach the maximum number of migration tries (line 9), it makes the backup resource and service graphs (lines 10–11) describing the current status of the computing system, and executes the migration process by the TRYMIGRATE method (line 12). This method, first, removes the migratable VNF from the original physical node, places it into the possible target node, and reconfigures the connected virtual links to use another physical link path between the VNFs. Furthermore, the method also maps the actual VNF *u* to the server from where the migratable one was moved, and determines the physical links to implement the virtual link between the *u* and *v*. So far, only the compute constraints of the VNFs have been checked, however, the network requirements may fail during the mapping.

The third phase of the migration procedure (lines 13–19) is to check whether the migration was successful. If the physical links fulfill the corresponding virtual link’s network requirements, then the TRYMIGRATE method returns true, thus the procedure of migrating was successful (line 13). Otherwise, RESETRG and RESETSGS restore the previous state of the resource/service graphs (lines 16–17), and continue with the next migration option from the list. Instead of checking all the possible migration options (migratable VNFs and possible physical nodes), in order to control the runtime, we test only a limited number of options. This iteration number can be controlled by defining the value of *max_try* and *max_vnf* environment values. The computational complexity of the algorithm is polynomial (details in Haja et al. (2018)).

In summary, the key innovation in DARK is the network-aware migration process that can migrate already deployed VNFs in order to make room for other services if it is possible. Our solution is able to migrate the VNFs either between two edge servers (that can be in the same or different clusters) or from edge server to the cloud. Naturally, this migration process takes all network requirements into account and the migration is triggered if and only if each requirement is met in the resulted state. In order to ensure the correctness of the DARK mapping, we use admission control: the elements of the SFC are deployed to the OpenStack servers if and only if the hosts contain enough computation (CPU, memory, storage) resource to store the VNFs and the network among the hosts fulfills network requirements (bandwidth and latency) which are defined between VNFs of the service. The admission control is ensured in every case, even when DARK migrates VNFs between clusters.

## 3.2. *MORCH: orchestration for multi-layer architecture*

Our second architecture proposal introduces an upper-level NFVO on top of VIMs. This is an adoption of our general orchestration system proposed for 5G networks (Vaishnavi et al., 2018; EU H2020 5G Exchange project) which supports arbitrary orchestration hierarchies where the orchestrators can control what to expose towards upper-layer NFVOs. We make use of the main elements related to resource orchestration and adjust them to edge cloud infrastructures and the requirements of future IoT applications. This approach is suitable for multi-provider scenarios. We note that the algorithms are described in our previous works (Németh et al., 2016; Sonkoly et al., 2018), however, it is important to inspect them as part of the overall system, as they might cause bottlenecks and crucially affect the performance. Therefore, here we focus on their integration into the architecture, address the algorithmic challenges of such adaptation, and highlight the key features required by the multi-provider operation. Consequently, we present the automated resolution of end-to-end constraints, domain-error handling based on a distributed backtracking mechanism and the inherent support of information hiding and resource aggregation.

### 3.2.1. *Proposed architecture*

The proposed architecture is shown in Fig. 5. Each edge domain or a cluster of edge domains is/are under the control of a dedicated VIM. On top of VIMs, a common northbound API is added which role is twofold. First, it exposes a bottom-up resource view including compute, memory

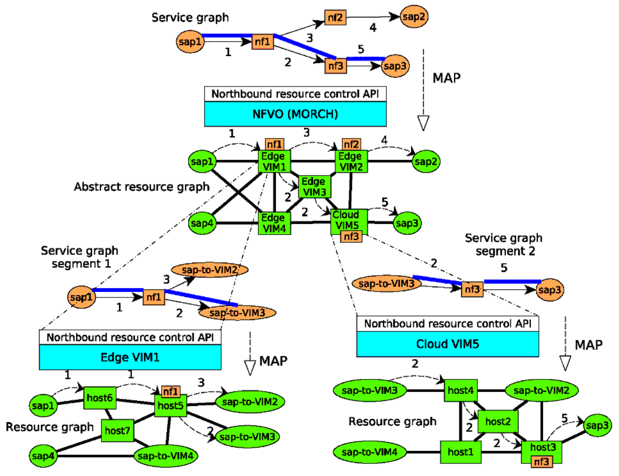


**Fig. 5.** Proposed multi-layer orchestration system for edge cloud infrastructures.

and storage capacity and a simplified network model. The abstract network model (big switch) describes ingress/egress link characteristics in terms of delay and bandwidth, and provides aggregated information on the internal network, i.e., path characteristics among ports (A similar model is presented in Sonkoly et al. (2015)). Second, the top-down resource control is also realized via this interface where the service requests as service function chains with latency and bandwidth constraints can be defined.

NFVO is responsible for integrating the underlying edge domains as well as the cloud resources. Central cloud or clouds is/are also controlled by dedicated VIM(s) with the previous API extension. The service requests received at the northbound API of the NFVO (same as the VIMs' API in our proposal), are mapped to underlying resource domains and decomposed accordingly. The "sub-service" requests are sent then to the involved edge or cloud orchestrators using the recurring resource control API.

### 3.2.2. *Highlights of the key algorithms*

The basic idea of our second orchestration algorithm for multi-layer scenarios (MORCH) is similar to the one explained earlier in Sec. 3.1 disregarding the inherent VNF migration capabilities. Multi-layer VNF migration can be achieved by re-running the MORCH algorithm when a dedicated monitoring system triggers it, which could result in relocating previously deployed VNFs. Our multi-layer orchestrator engine uses a graph-based, heuristic-guided greedy backtracking search on the resource graph structure. In our earlier work, we have presented the details of the algorithm (Németh et al., 2016; Sonkoly et al., 2018) and its good scaling properties, now we show how it has been adapted to create our multi-layer orchestration system. MORCH operates on the abstraction level of the substrate topology infrastructure which is shown by the underlying VIMs and their interconnections.

An elementary embedding step of MORCH is the greedy mapping of a VNF and an adjacent service graph link onto a hosting (abstract) big switch node (shown by the VIMs, respectively) and onto a path consisting of VIM connections. After each successful greedy step, the algorithm's data structures representing the currently available substrate resources are updated. In case such a greedy step is not able to find a suitable hosting option, MORCH performs backtracking similarly to DARK. The orchestration engine provides an embedding solution when all elements of the service graph have been successfully mapped respecting each aspect of the requirements or refuses the request.

Our MORCH algorithm is parameterizable both in terms of search space size and search behavior. The former can be tuned by the backtracking parameters (*i*) defining how many hosting alternatives of an elementary step shall be stored (branching factor), and (*ii*) how many consecutive greedy steps can be undone in the search tree (backtracking depth). Search behavior is controlled by several parameters of the preference function defining the heuristic. At each greedy step, the most preferred substrate node and path are chosen as the host of the currently considered VNF and its adjacent service graph link.

Due to the hidden information shown by the abstract big switch nodes of the VIM layer, it is possible that a selected embedding solution on the abstract view of an upper layer turns out to be impossible to map on a VIM's full infrastructure view. In this case, MORCH and the domain orchestrators communicate this failure to the appropriate domains, which undo the failed service instantiation. A subset of the underlying domains might have successfully mapped their part of the segmented service graph, in case of a failure, these instantiations must be undone. Our algorithm supports this scenario efficiently: if a lower abstraction layer failure notification arrives, the greedy backtracking search of MORCH continues from the latest solution, eliminating the need to start the whole orchestration process from scratch.

In addition to basic service graph requirements, such as node and link capacities, VNF type constraints and link-wise delay requirements, MORCH supports end-to-end delay constraints on service graph paths between SAPs. An end-to-end path is shown in Fig. 5 on service graph links 1, 2 and 5. If such an end-to-end requirement is given for orchestration over an abstract resource view, a delay budget is allocated (respecting the overall end-to-end requirement) for all affected abstract VIM nodes, whose orchestrators over lower level resource abstractions receive this delay budget as input end-to-end requirements between their inter-domain endpoints (SAPs). See the input service graph segments for Edge VIM1 and Cloud VIM5 in Fig. 5 on their respective service graph segments. This approach provides the multi-layer support of guaranteed end-to-end path delays over a unified abstract interface.

## 4. Proof of concept prototypes

We have implemented both architecture options as proof-of-concept prototypes. The main components and relevant implementation details are summarized in this section.

### 4.1. Extended OpenStack

As today's one of the most widely deployed open-source VIM is OpenStack, we target that platform. Making use of the previously presented DARK algorithm (see Sec. 3.1), we extend OpenStack's scheduler algorithm to turn it into a network-aware resource orchestrator. The key features of this approach are also demonstrated in Szalay et al. (2019). In addition, the code has been released as open-source (DARK).

On the one hand, OpenStack is a cloud computing platform, used as the *operating system* of both private and public clouds. It provides Infrastructure as a Service (IaaS) and it is responsible for the management of large pools of compute, storage and networking resources. Many loosely coupled components are developed as independent projects where the components are communicating with each other through well-defined REST APIs (Logical architecture of Openstack). On the other hand, our DARK algorithm maps service function chains to an internal topology, which defines the exact resources where the virtual service components should be executed. As a proof-of-concept, we replace the default *nova-scheduler* of OpenStack with our own algorithm in order to support network-aware VNF placement. This prototype is also able to run automated measurements to determine physical network characteristics. In this section, the key elements and implementation challenges are described.

An example setup is shown in Fig. 6. All compute nodes are controlled by the same OpenStack controller. DARK creates an abstract model of the physical infrastructure, which contains a delay matrix and a resource graph. The delay matrix defines delay between all node pairs. The resource graph includes all infrastructure components, i.e., nodes, links, clusters. The orchestrator algorithm maintains this abstract model and processes the incoming service requests.

#### 4.1.1. Network status measurement

Since currently OpenStack does not provide any network related metrics, it cannot take them into account during the orchestration process. To solve this problem, we implemented a measurement method which is conform with our previously introduced network model. Our tool is based on VMTP (VMTP) which is a data path performance measurement module for OpenStack. It performs automatic measurements between the different virtual networks, but can also be used to benchmark native hosts. It connects to the given nodes via SSH, executes the measurements by using the selected protocols (TCP, UDP, ICMP), then returns the result to the management server.

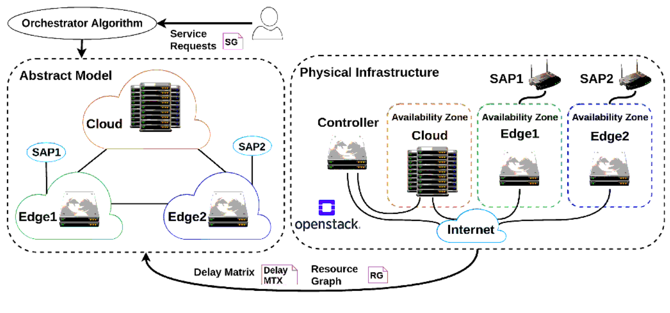


**Fig. 6.** Orchestrating OpenStack resources with DARK.

Each OpenStack compute host in our reference cloud is configured to belong to one custom Availability Zone. An Availability Zone may represent an edge cluster or a cloud according to our terminology. As we assume the servers located in the same cluster are deployed physically close to each other (e.g., in the same rack), it is enough to measure the latency and bandwidth values between 1 and 1 randomly selected servers in each zone. By applying this method we can construct the delay and bandwidth matrices that describe the parameters of the underlying physical network.

Furthermore, through the OpenStack API we also collect the available compute node resources (CPU, RAM, storage) from each hypervisor. From the gathered information, we can build up the resource graph that contains compute resources extended with the networking related features.

#### 4.1.2. OpenStack scheduler algorithm and modifications

OpenStack's physical resource orchestrator component is Nova. It uses its own filter scheduler for filtering and weighting in order to make informed decisions where a new instance should be created. During the VM placement *nova-scheduler* iterates over all compute nodes, evaluates each of them against a set of filters. The list of resulting host is sorted by the administrator-defined weights. This default filtering operation cannot deploy our virtual services properly because there is no standard filter class that tackles the network resources (delay, bandwidth) between infrastructure nodes. Although with Nova API it is possible to deploy a VM on a manually specified host. In our prototype, we use this feature for deploying the VNFs on the host given by our resource orchestrator algorithm.

The next step in the prototype's workflow is ensuring correct traffic steering. OpenStack officially supports service chaining since its release Pike, i.e., DARK can be applied since 2017. Network traffic steering with Neutron port chains is provided by the *networking-sfc* (OpenStack Service Function Chaining) module. Our code can use Neutron API to create an ingress and an egress port for each VM. These ports are grouped into port pairs by the owner VM. The port pairs are grouped into Neutron port pair groups by the virtual link connections. A port chain consists of a set of Neutron port pair groups to define the sequence of service functions. These Neutron objects make it possible to deploy our traffic steering model for service chaining that uses only Neutron ports.

#### 4.1.3. Scheduling SFCs with OpenStack Heat

Heat service is the main orchestration service in OpenStack. It implements an upper layer engine to launch multiple virtual resources based on templates in the form of text files. In a Heat Orchestration Template one can easily define resources to be deployed. Templates may also describe the relationships between resources, e.g., which port is connected to which instance. This enables Heat to call the proper OpenStack service APIs to create all resources in the correct order to launch an application.

Besides the direct Nova and Neutron API calls, DARK is able to use Heat API. We convert our SFCs into Heat templates where each component in the chain is defined as Heat resource such as instance, port-pair, port-pair-group, port-chain. In that case if a VNF instance is already present in OpenStack, DARK does not deploy the instance again, it invokes the instance with its id in the template.

#### 4.1.4. Challenges and limitations of OpenStack

One of the limitations of OpenStack stems from the heavy-weight virtualization technique used when VNFs are implemented as VMs. Our single provider setups based on DARK use OpenStack to manage

VNFs as VMs in both edge clusters and in cloud data centers. Therefore, the deployment time of the services and VNF startup times are expected to be much larger compared to other scenarios using software containers. However, virtual machines can provide better isolation among the tenants.

If the latency is too high between the controller and the compute nodes, then the controller cannot execute the commands properly on compute nodes. We conducted several experiments with emulated delay between the nodes in order to reveal the performance characteristics in such scenarios. The experiments confirmed that OpenStack compute service is able to work in distributed environments when we have non-negligible delays among the nodes. We could successfully deploy VMs in extreme scenarios where the latency between the controller and the compute node was 10 s. Of course, the spawning method, i.e., starting VM into active state, took a few minutes.

Our orchestrator algorithm in DARK considers the possibility of migrating VNFs between compute nodes. Therefore, we implemented the migration API calls in our prototype code. Migrating VMs between compute nodes is not trivial if there are different types of CPUs in the compute nodes. In order to ensure that migration works correctly, we have to make some slight modifications in Nova's configuration files to solve the issue. For instance, we set virtualization type to Qemu instead of KVM on all compute servers. Note that to the best of our knowledge, there are no solutions to live migrate VMs between different OpenStack clouds, another important reason for federating computing clouds and edge clusters under one common VIM. Furthermore, as only one controller is needed in this case, our proposed setup does not take hardware resources from the compute nodes in the edge domains with limited capacity.

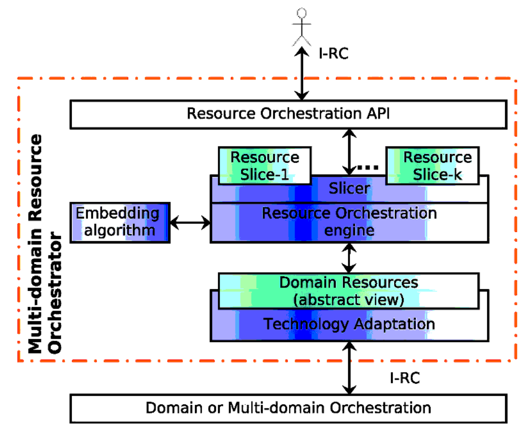
### 4.2. *Multi-layer orchestration system*

In Vaishnavi et al. (2018) and EU H2020 5G Exchange project, we proposed a general purpose multi-domain orchestration system for future 5G networks. Here, we adopt the main elements related to resource orchestration and adjust those to IoT applications and edge cloud infrastructures.

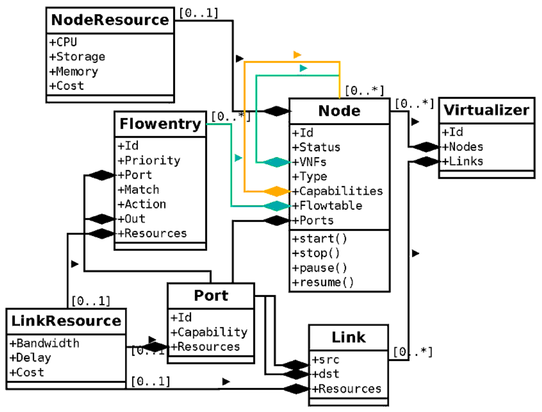
#### 4.2.1. *Multi-domain resource orchestrator*

Our Resource Orchestrator (RO) encompasses and coordinates multiple components as it is shown in Fig. 7a. In general, it is responsible for exposing different virtual resource views upwards and satisfying service deployment requests. The requests are expressed on the high-level virtual views (Resource Slices) and mapped onto the full domain view which encompasses the underlying resources and topologies. In Fig. 7a, green boxes correspond to resource views, while red boxes indicate orchestration or control related elements. During the orchestration workflow, RO engine invokes the embedding algorithm module, MORCH, which performs the mapping of the service requests to the available resources according to the configured policies. The result describing the full deployment is then sent to the Technology Adaptation component. It provides a domain-agnostic resource abstraction and virtualization for different resources, technologies or administrative domains. The recurring resource control interface is denoted by I-RC, which is built on the information model presented in Sec. 4.2.2. As we use the same interface at north and south, a recursive orchestration hierarchy can be constructed. The underlying entity can be either a domain orchestrator or another multi-domain orchestrator aggregating different domains.

Slicer is an integrated part of the RO and its role is threefold: *i*) it introduces multi-tenancy by configurable northbound views corresponding to consumers; *ii*) it enforces policies with regards to slice to resource mapping, e.g., if a consumer is limited to a pool of domain resources, then these attributes are set before calling the embedding function; *iii*) it enforces operational policies with respect to consumer-to-consumer sharing of service instances.


I-RC
Resource Orchestration API
Resource Slice-1
...
Resource Slice-k
Slicer
Embedding algorithm
Resource Orchestration engine
Domain Resources (abstract view)
Technology Adaptation
Multi-domain Resource Orchestrator
I-RC
Domain or Multi-domain Orchestration

(a) Software architecture of our orchestrator



(b) Information model: simplified view

**Fig. 7.** Software architecture and information model of our proof-of-concept multi-domain orchestrator.

We have implemented our embedding algorithm, MORCH, with all of its features which are presented in Sec. 3.2. MORCH works on the abstract resource view, which is gathered from the underlying orchestrators and constructed on-the-fly (For example, in Fig. 5, NFVO constructs the abstract resource model based on the information gathered from e.g., Edge VIM1 and Cloud VIM5). In case a failure occurs in the orchestration procedure on any abstraction layer, the embedding algorithm supports the framework by finding an alternative solution on its resource view respecting all of the original service requirements. This trial-and-error mechanism concludes by receiving and propagating positive service embedding results from the lowest (physical) layer orchestrators.

#### 4.2.2. *Information model and the resource control API*

Our information model is a central element which is used at the recurring resource control interfaces (I-RC). This model enables the multi-layer (and recursive) operation by abstracting both *i*) the bottom-up network of compute resources and function capabilities, and *ii*) the top-down view of control over the virtualized infrastructure. From general aspects, our model is similar to ETSI's NFV MANO data models (Network Functions Virtualisation, 2014), however, it extends that in multiple ways in order to enable multi-operator scenarios. For example, our model supports the abstraction of an arbitrary topology of resources and capabilities, in addition, we introduced the notion of typed VNFs, and our model allows full recursion, i.e., the northbound and southbound representations are the same for resource orchestrator components.

We designed an object-oriented information model, a simplified version of which is shown in Fig. 7b. On the one hand, the model is capable of describing an arbitrary topology of resources and capabilities. This information corresponds to the resource view exposed by an orchestrator towards an upper level entity. On the other hand, services (more specifically, service function chains) can be formed as allocation requests defined on the given (northbound) resource view. The task of service embedding in any given layer is analogous to match the resource requests from the northbound topology to the southbound topology of resources and capabilities.

A *node* represents either an abstraction of node resources and capabilities or a VNF allocated on a node. The former type of nodes is referred to as Big Switch with Big Software (BiS-BiS). Node objects have *ports* representing connection points; *links* connecting ports of different nodes define abstracted physical interconnection; links interconnecting ports of the same node, i.e., internal links, capture the aggregation for a topology, e.g., if a domain of 10 MPLS switches are aggregated into a single node, then the external ports of the MPLS domain will appear in the abstract node with internal links that characterize the edge-to-edge LSPs, like latency, bandwidth, QoS class, etc. In order to allow forwarding control for nodes, flow entries can be defined: we use *port - match - action* sets following the SDN design principle, but we support various technology specific mappings, as well. The flow rules may include *i*) matching of input port, abstract tags or other technology specific header fields, *ii*) actions such as output to port, push/pop abstract tags or any other technology specific packet manipulation. Basic life-cycle management operations are contained in the *status* field of each node, such as create, start, stop, and pause. BiS-BiS nodes can be connected to each other representing direct or logical connectivity between the corresponding ports. Service Access Points (SAPs) represent external connections where customers can be attached to the system.

### *4.2.3. OpenStack Domain Orchestrator*

In order to evaluate our concept in realistic scenarios, the adaption to today's VIMs is crucial. We have developed a dedicated library released as open-source (UNIFY virtualizer library) supporting the implementation of I-RC interface onto different VIMs.

As OpenStack is the most widely used open-source cloud "operating system", we integrated it in our framework. In Fig. 8, two OpenStack Domain Orchestrators (ODO) are shown with their main components and their managed OpenStack environments. ODO exposes a northbound interface, which is used to interact with upper layer orchestrators following the information model presented in Sec. 4.2.2. The resource orchestrator module parses the given configuration files, maintains the current topology and the database of supported VNFs. The adapter library provides the necessary modules and helper functions. ODO manages OpenStack via its REST API.

ODO currently supports two operation modes. Fig. 8 shows the differences between these modes, and also the connection between two OpenStack domains with different modes. The first one is an SDN compatible mode, which can handle requests containing VNFs connected by SDN flow rules. In Fig. 8 the domain on the left hand side represents this SDN compatible operation. The key requirement inside an SDN network that all of the VNFs should contain a wrapper function with an included Open vSwitch. This wrapper module allows handling control plane messages coming from the orchestrator destined to the VNF. Furthermore, the wrapper creates VXLAN tunnel endpoints and virtual interfaces in the deployed instances. In our implementation, all of the VNFs running in this scenario use a shared Neutron network, while their traffic is separated by the VXLAN tunnels. Using the SDN compatible operation mode, we need a special VM called "central_VM", which is responsible for handling internal and external data plane connections. All of the data plane flows go through the central_VM, where the traffic is aggregated by Open vSwitch flow rules. When the ODO starts, it checks whether there is any central_VM running in the managed domain.

The other operation type of ODO supports only "legacy IP" network connections. In Fig. 8 the domain on the right hand side represents this kind of operation, where the VNFs are deployed as simple VMs with no extra functions. When the orchestrator deploys a VM in this type of domain, it creates as many Neutron networks as required to fulfill the VNF's interface constraints. Each Neutron network is connected to at least one router to provide the connection between the deployed VNFs.

As shown in Fig. 8, we have implemented a solution to interconnect legacy IP domains with SDN domains. SDN traffic is represented with light blue color, while legacy IP traffic is depicted with grey. This solution assumes BGP-based IP VPN networks between domains realized by the BagPipe driver. The BaGPipe driver for the BGPVPN service is designed to work together with the Open vSwitch ML2 mechanism driver. It relies on the use of the bagpipe-bgp BGP VPN implementation on all compute nodes and the MPLS compatibility of Open vSwitch. BGP VPN is deployed and managed by domain operators, in particular to manage Route Target identifiers that control the traffic isolation between different VPN networks. In our multi-domain system, for the referred interconnection, both OpenStack domains must have Neutron routers associated to at least one BGP VPN network. In the SDN domain, there is a special VM called proxy_VM, which makes the traffic encapsulation and decapsulation between the two networks. On the SDN side, the proxy_VM has VXLAN tunnel endpoints, while on the legacy IP - BGP VPN side, this instance is connected to at least one router associated to the BGP VPN network.

### *4.2.4. Docker Domain Orchestrator*

Docker is another important VIM to be adapted, which manages light-weight containers instead of VMs. The architecture of our implemented Docker Domain Orchestrator (DDO) is shown in Fig. 9. The high-level software architecture is similar to the previous one, however there are some key differences in the implementation. DDO uses a slicer module to simultaneously manage multiple resource and network slices.

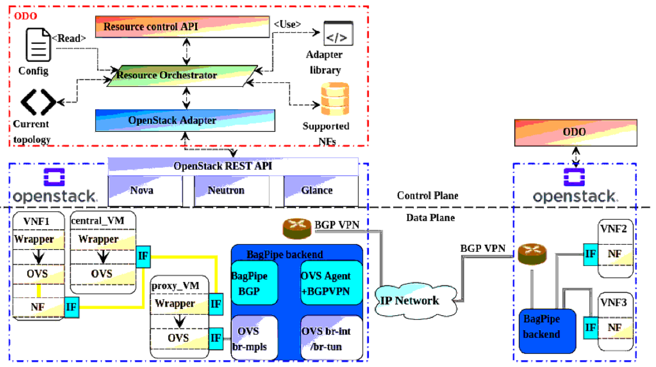


**Fig. 8.** OpenStack Domain Orchestrator (ODO): connecting IP and SDN-compatible OpenStack domains.

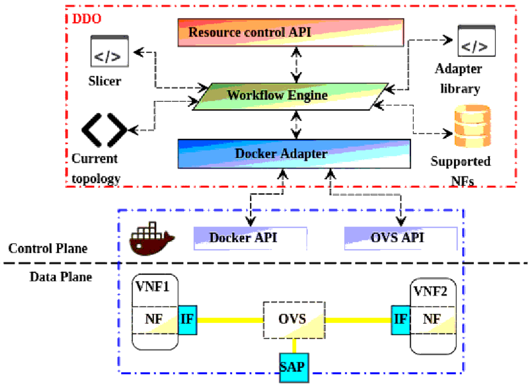


**Fig. 9.** The architecture of Docker Domain Orchestrator.

Obviously, the domain adapter uses different API calls on Docker domain, than the ODO uses on OpenStack. The Docker domains have the same number of Open vSwitch as the number of created slices. These Open vSwitches handle the SDN network flows inside the domains, which are realized by VXLAN tunnels, like in our OpenStack solution. The Docker API calls are responsible for the containerized VNF deployment.

#### *4.2.5. SDN and legacy IP Network domains*

We implemented an SDN Orchestrator (SDNO), which is capable of managing traffic forwarding rules in OpenFlow networks, without the generic capability to run VNFs. However, functions that can be realized with OpenFlow rules, e.g., a traffic splitter/duplicator or an IP router, can be placed into such a domain. We use SDNO both in virtual (Open vSwitch) and hardware based OpenFlow domains.

We use technology specific adaptation to implement end-to-end connectivity over domains with different network service capabilities. Such adaptations are implemented in a network service specific orchestration component. For example, for IP VPN across IP/MPLS and SDN networks, the IP specific component manages IP/MPLS BGP VPN specific parameters, configures static routes over the SDN domain and stitches the two domains together by injecting routing information of the SDN domain into BGP at the stitching point.

## 5. Evaluation

In this section, we evaluate the performance and scalability characteristics of our proposed orchestration systems. First, we define a common set of experiments following the general structure of the envisioned application presented in Sec. 2. Second, the scalability properties of the embedding engines and the overall orchestration processes are analyzed in terms of the number of clients and operated edge domains. We focus on control plane operations as the behavior of the data plane components is independent from the orchestration system.

### *5.1. Description of the experiments*

In order to conduct similar experiments with the two distinct orchestration systems, a common set of scenarios are defined in a general format. Here, we use the BiS-BiS representation introduced in Sec. 4.2.2 to describe topologies, network and cloud resources and also the service requests. These data models are converted and adjusted to the input formats required by the orchestrators, respectively.

The bottom part of Fig. 10 shows the general structure of our distributed cloud (and edge) resources and the topology connecting them. The core network, which consists of four transport nodes

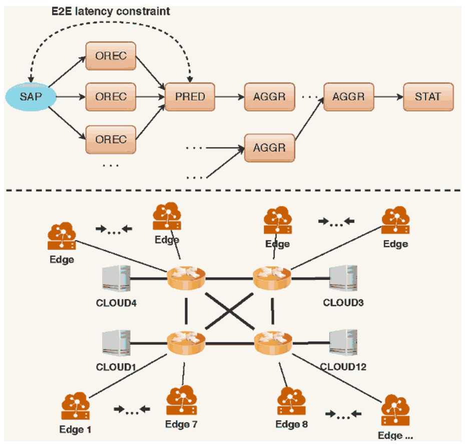


**Fig. 10.** Investigated scenarios: test topologies with cloud and edge resources (bottom), service requests (top).

(switches) connected in full-mesh, is responsible for connecting the central data center nodes with an arbitrary number of edge domains. The available cloud resources (CPU and memory), which could represent either private or public cloud resources, are divided into four BiS-BiS nodes, each of them is connected to an associated core switch. Edge domains are also represented as standalone BiS-BiS nodes with limited amount of resources and with dedicated SAP access points, respectively. The total number of edge domains is determined by the given test scenario and they are uniquely distributed among the core switches. Each virtual link is characterized by its latency and bandwidth properties (generated randomly), while all the resource nodes additionally specify the types of supported VNFs, i.e., the set of VNFs which can be mapped on and deployed into the related domains.

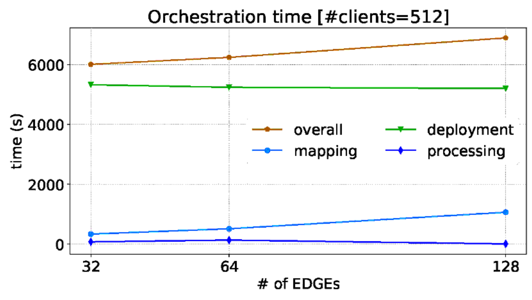
Our service requests used in the experiments are illustrated in the upper part of Fig. 10. Generated services consist of chains of connected VNFs, which provide the alerting functionality for one car/client. Each VNF specifies the type of realized functionality and the required amount of CPU and memory, while the virtual links in the service chains can define optional latency constraints (Furthermore, end-to-end latency requirements can also be specified for arbitrary paths in the graph). In our experiments, OREC (and optionally PRED) functions are restricted to be placed close to the clients/SAPs at the edge of the topology. In one service chain there is only a single instance of PRED and STAT VNFs but the number of OREC functions is randomized within a predefined range. AGGR functions along with the single STAT are shared between the chains forming a balanced tree which size fits to the number of chains. The service chains in one test request are uniformly distributed among the edge SAPs.

To perform large scale control plane experiments, we generated multiple service requests along with the related resource topology for each test scenario. The setups are based on increasing number of edge domains and service chains, which correspond to the number of wireless transmitters at the network edge and to the number of clients/cars that need to be served. As we focus on control plane operations, in most cases, the data plane of the underlying domains and VIMs are emulated. However, the deployment characteristics of our VIM implementations, such as Docker and OpenStack, are also analyzed based on small scale tests investigating the overhead of VNF launching and the configuration of traffic steering rules.

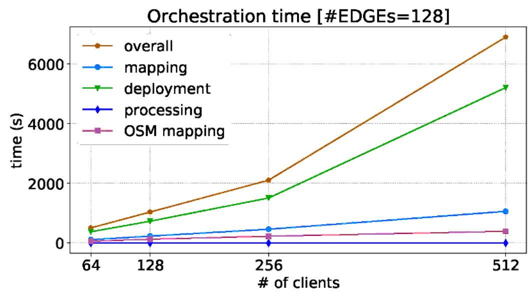
As Open Source MANO (OSM) (Open Source MANO) is one of the most prominent, production-quality orchestration system for NFV-based network services and it is released as open source, we use it as a baseline in our performance evaluation. However, OSM does not support any delay requirements in the service description and it does not take the latency characteristics of the underlying networks into account, its mapping procedure can be considered as a baseline for the control plane operation. Therefore, we constructed the simplified version of our service requests described in OSM's format, i.e., the complexity (number of constituent VNFs) and the structure (links among the VNFs) of the services are the same but the delay constraints are excluded, and we conducted experiments with OSM in the same environment. Here, we focus on the mapping phase of OSM's operation because in this scenario, its deployment engine calls the same OpenStack APIs which are used by our prototypes, thus the deployment times are similar.

### *5.2. OpenStack experiments with DARK extension*

We set up a dedicated, tailor-made OpenStack cluster which supports large scale experiments mainly addressing the control plane performance. Each compute node and the controller are operated within distinct Docker containers spread across three physical servers. A dedicated server hosts the controller node container, while on the other two machines, 65 compute node containers are launched per server as central cloud nodes or edge nodes. The latency and bandwidth parameters are emulated in our environment by `tc` (Linux Traffic Control). Each server has the following technical specification: Intel(R) Xeon(R) CPU

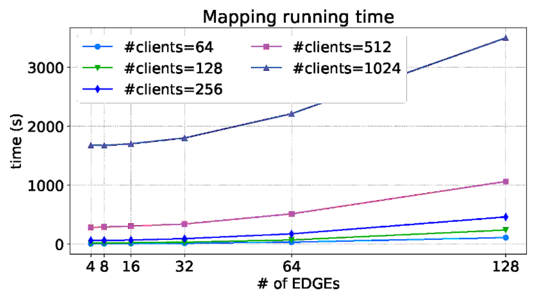


(a) Orchestration times of a service including 512 clients (cars) on different topologies

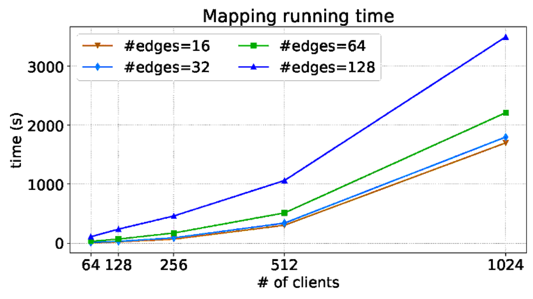


(b) Orchestration times on a network with 128 edge clusters

**Fig. 11.** Detailed DARK orchestration times of large scale experiments in terms of increasing number of edge clusters (top) and served cars (bottom).



(a) Performance of DARK with increasing number of edge clusters



(b) Performance of DARK with increasing number of served clients

**Fig. 12.** Scalability characteristics of the DARK mapping algorithm.

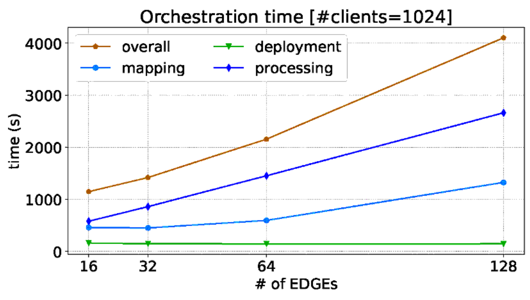
E5-2620 0 @ 2.00 GHz, 62 GB RAM. All of the compute nodes are deployed with fake Nova drivers, which means that VM actions, such as creation, launch, stop or getting diagnostic information, bypass the hypervisor which manages only the control plane messages. In order to enable experiments in such scale, several tweaks had to be applied. First, we had to increase the number of allowed connections to the MySQL server because some of the experiments contain thousands of service requests. OpenStack stores all information on its components (including VMs, ports, port-chains, etc.) in the controller node's MySQL database which is maintained by several internal services. One can easily see that the number of database connections is directly proportional to the number of instantiated components. Second, a similar approach was required to guarantee the proper operation of the message queue handler, i.e., RabbitMQ, and the maximum number of open file descriptors had to be increased.

During the experiments, DARK was running on the same physical host as the OpenStack controller. DARK operates on top of the regular OpenStack services and here, we used the Heat API to physically realize the calculated deployments. As DARK operates with chains as service request inputs (see Sec.3.1), first we transform the test requests to chains in our experiments. During the transformation we iterate through all the paths in the original request starting from the SAPs. As the transformation is finished, all the SFCs contain all the original service links once. A VNF may appear multiple times in the SFC set, however, DARK algorithm does not deploy the VNF again but it deploys only the unprocessed link between its two endpoints.

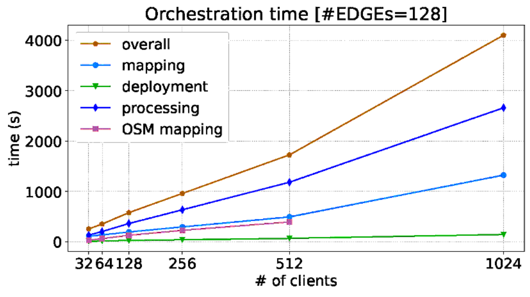
Fig. 11 presents the *overall* orchestration times of the service deployments with extended OpenStack for a selected number of served cars and managed edge clusters. In the figures, the runtimes of the different orchestration steps (mapping, processing, deployment) are shown in terms of increasing number of edge clusters (Fig. 11a) and increasing number of clients (Fig. 11b), respectively. We opt to use exponentially increasing number of edges and clients in order to cover a wide range of scenarios with a feasible number of experiments. *Mapping* is the key process of the orchestration invoking the DARK algorithm, while the *processing* time includes the transformation of the result of the mapping algorithm to the corresponding Heat template. The Heat service sends a callback message when it verified the request and sends the proper low-level requests, e.g., VM instantiation, port creation, port-chain creation, to other OpenStack services. The *deployment* time gives the elapsed time between the template transformation and the time when the last component of the SFC is created.

The results confirm the good scaling characteristics of the mapping algorithm. Both in network size and number of clients, the computational complexity is polynomial (near to linear in the analyzed range) which stems from the design of our simple greedy heuristic. Even in case of the extreme scenario including 512 clients and 128 edge domains, the runtime of the mapping algorithm is around 1000s and the overall orchestration time is mainly determined by the deployment phase (around 5000s). This holds for all other experiments, i.e., the entire time needed to deploy the SFCs is mostly affected by the internal operations of OpenStack. In addition, the processing time can always be ignored comparing to other phases as the conversion between the data structures is not a complex task. As a baseline, the performance of the mapping process of OSM is also shown in Fig. 11b. Obviously, DARK needs more time to solve a more complex mapping task (taking also the delay constraints into consideration) but its performance is comparable to the baseline.

More realistic operation regimes are evaluated based on simulations with the algorithm (without OpenStack) and the main results are shown in Fig. 12. Here, only the running time of the mapping algorithm is evaluated for a diverse set of scenarios. We present the mapping time results with increasing number of edge clusters in Fig. 12a and with increasing number of clients in Fig. 12b. As both plots of Fig. 12 show, the number of edge clusters has less effect on the runtime of the algorithm than the number of clients. This behavior can be explained by the online operation of DARK, which means that the service requests are received and processed sequentially. Therefore, DARK cannot place each component of the full service in one step rather a gradual embedding method is realized. In case of large requests, with the increasing number of already deployed service functions and VMs, the chance of migration

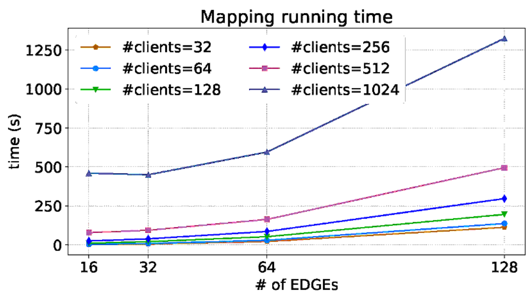


(a) Orchestration times of a service including 1024 clients (cars) on different topologies

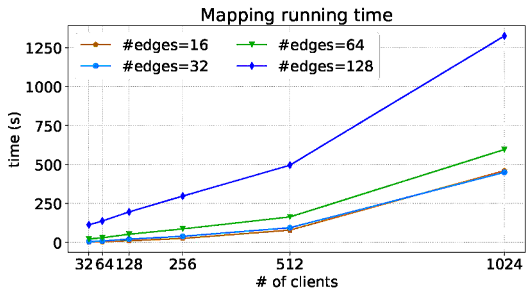


(b) Orchestration times on a network with 128 edge domains

**Fig. 13.** Detailed orchestration times of large scale experiments in terms of increasing number of edge domains (top) and served cars (bottom).



(a) Performance of MORCH with increasing number of edge clusters



(b) Performance of MORCH with increasing number of served clients

**Fig. 14.** Scalability characteristics of MORCH mapping algorithm.

also increases. Migration is one of the most time-consuming operation phase and it gets worse when the topology becomes saturated which results in increased overall runtime. In Fig. 12a, a relatively poor performance can be observed for the use-cases with 1024 clients. For example, if we have 128 edge clusters in the underlying infrastructure, the orchestration time is around 3000s, which seems unsatisfactory for the first sight. However, DARK received more than 3000 service requests in this scenario, thus the average runtime for a service request is less than 1s, which we consider acceptable. It is worth noting that in the presented scenarios, we fed the system with a single batch request containing all users' services. It can be considered as a worst-case scenario because in a more realistic operation mode, customers send their requests after each other distributed in time. Therefore, the per-user processing time will be under a few seconds and it is not necessary to wait till the end of the deployment of all others' service components.

The good scaling properties (polynomial complexity) of DARK are confirmed by the experiments and even for the most complex setup, including 1024 clients and around 3800 VNFs, the average mapping time of one user's service remains under 1s. Although the total mapping time seems large in Figs. 12 and 11 indicates that the bottleneck of realizing complex services, encompassing an extreme number of VNFs, would be the deployment process of OpenStack. Figs. 11 and 12 also show that the number of clients and the complexity of the services (number of VNFs) have higher impact on the deployment time, including both the mapping and deployment phases. We can conclude that DARK can be a valid option for single-provider setups serving a few hundreds of customers over an infrastructure including even 100 edge domains.

### 5.3. Multi-layer orchestration system based on MORCH

In our multi-layer experiments, we address similar resource domains encompassing the same amount of compute and network capacity as it was configured for DARK scenarios. The main difference here is in the control plane, more specifically, a dedicated control plane hierarchy is constructed on top of the lower layer resources as it is presented in Sec. 3.2. On the one hand, we have a top-level multi-domain resource orchestrator running MORCH. On the other hand, domain orchestrators are in charge of controlling lower level compute and network resources and exposing abstract resource views towards the upper level multi-domain orchestrator. Instead of real domain orchestrators (presented in Sec. 4.2.3 and Sec. 4.2.4), we use an emulator implementing the same northbound API and the same control plane operations (Actually, the emulator inherited the software modules from ODO and DDO). By these means, the performance of the multi-layer control plane workflows and API overheads can be evaluated. The behaviors of the implemented domain orchestrators are highlighted briefly at the end of the section.

In the analyzed test scenarios, we had a multi-domain orchestrator, an SDNO controlling the core network (including 4 switches), 4 instances of ODO managing the central cloud domains, and configurable number of DDOs managing the distinct edge domains. The test cases were conducted on platform with Intel Xeon CPU E5-2640 v3 @ 2.60 GHz, 16 GB RAM and each software instance was pinned to one vCPU core to avoid undesired race conditions.

Fig. 13 presents the overall orchestration times and relevant phases of the operations measured at the multi-domain orchestrator for a selected number of served cars and managed edge domains. In Fig. 13a, the scalability characteristics of the different orchestration steps are shown in terms of increasing number of edge domains, while Fig. 13b presents the same results in the dimension of served clients. The plots show that even in case of 100+ edge networks and 1000+ clients, the pure mapping time remains considerably under the time of additional processing and conversion tasks, such as calculating the deployable part of the orchestrated service for a managed domain, converting the internal representation into the appropriate data exchange format and other domain provisioning tasks. Moreover, the baseline experiments with OSM also confirm the efficient operation of the mapping algorithm of MORCH. Our algorithm, solving a more complex task than OSM, exhibits almost the same performance as OSM's mapping engine (The largest scenario is rejected by OSM because it exceeds the maximum service size allowed by the system). Here, the deployment time contains only the VIM's control plane overhead gained from our emulator (without real data plane operations) and the delay caused by the registration of the deployment results at the multi-domain orchestrator's side. The results show that the control plane overhead at the VIMs added by our novel northbound API is negligible compared to the mapping and processing phases. The mapping time and also the overall orchestration time show polynomial scaling characteristics both in the number of clients and edge domains which are in line with the analytical results calculated for the general algorithm in Sonkoly et al. (2018).

Fig. 14 depicts the detailed evaluation of the runtime of the core embedding algorithm for a wide range of relevant scenarios based on simulations (without underlying domain orchestrators). The good scaling properties of the algorithm are confirmed for the analyzed range. The test results also show that the mean runtimes, up to 32 edges and up to 512 clients, are relatively close to each other, anticipating efficient operation. Slight randomization in service chains introduced minor difference in running times as the measured relative standard deviations fell below 4% in all performed test cases.

Finally, the performance of the domain orchestrators is investigated focusing on the comparison of the regular operation and the overhead introduced by our northbound API extension. As an example, a given set of scenarios is picked up from the previous multi-domain experiments (topology consisting of 32 edge domains) and the behavior of a selected Docker Domain Orchestrator is analyzed. Here, we repeated the same experiments but one of the emulators was replaced by a DDO instance with configured data plane (with physical interface bindings and VXLAN-based virtual links). Fig. 15 shows the impact of increasing number of clients on the deployment time (plotted in log scale). The

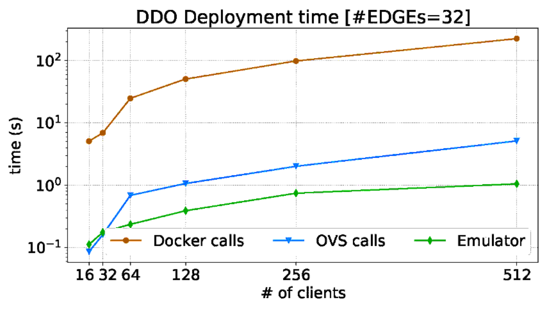


**Fig. 15.** Performance of a single Docker Domain Orchestrator compared to the pure control plane operations implemented by the emulator.

number of clients indicates the size of the original request sent to the top level orchestrator. As the clients are distributed randomly among the edge domains, the targeted DDO served only 1/32 portion of the customers in average. Fig. 15 depicts two phases of DDO operation and as a reference, plots the performance of our emulator (green curve) indicating the pure control plane operation. The blue curve corresponds to Docker-related tasks including the communication overhead of the Docker daemon, the container initialization time and also our control plane extensions. While the orange curve shows the flow rule insertion time required to configure the appropriate traffic steering rules. The results indicate that the overhead introduced by our northbound API extension is smaller in orders of magnitudes than the basic operations of Docker. More specifically, the deployment time is mainly determined by the VIM itself (Docker-related functions).

Our other domain orchestrator, namely ODO, manages OpenStack. As it is a much more complex VIM than Docker, we expect that the overhead caused by our API extension is not significant comparing to the general operations. We confirm this expectation via simple experiments which are capable of providing a good insight into the system's behavior. In Table 1, the runtimes of two different operational phases (and the overall performance) are shown for two different service

**Table 1**
Performance of ODO: simple experiments.

| VNFs in the service | Control plane operation (sec) | Deployment time (sec) | Overall time (sec) |
|---|---|---|---|
| 1 VNF | 12.65 | 79.49 | 92.62 |
| 10 VNF | 86.16 | 807.2 | 893.4 |

requests. The first phase corresponding to our control plane API implementation lasts till the VMs are requested from OpenStack. The second phase, which is responsible for the deployment, includes the pure VM instantiation and the OVS API calls to the wrapper functions (Here, we use ODO in SDN compatibility mode). One can see that according to the expectations, the deployment times are the significant components mainly affecting the overall performance for both scenarios. More information and further experiments with ODO can be found in Gerő et al. (2017).

### 5.4. Discussion

The ultimate goals of the two proposed solutions are similar, however, for different scenarios, different options can be a better fit. Here, we summarize the pros and cons of our proposals and compare them from multiple aspects.

On the one hand, OpenStack with DARK controller is a simple system and could be a good choice for an operator owning all the resources in a smaller domain. If she has an operational OpenStack with central cloud resources, then extending it with novel edge domains is quite trivial: additional compute nodes should be configured on-site and connected to the center. In addition, DARK controller has to be installed, e.g., on the master node. This solution scales well for a moderate number of edge nodes, however, when a service request consists of too many VNFs (more than 1000), the deployment time can be unacceptable. This stems from the internal operation of OpenStack. If we add new services gradually and not in "batch mode", this issue can be mitigated.

On the other hand, the multi-layer orchestration system introduces overhead in certain dimensions. First, the available VIMs have to be extended with a novel northbound API and the corresponding control plane mechanisms. Moreover, the same API must be used in all domains. Second, a dedicated component, i.e., the multi-domain resource orchestrator, has to be installed and configured in the system. If multiple providers are involved in the service provisioning, this approach has to be applied. However, this is a feasible option for single operator scenarios, as well. Based on the inherent service decomposition and distributed operation, the scalability characteristics of the multi-layer orchestration system are much better in terms of network size and service complexity. In case of large networks with thousands of users or global operators, this is the reasonable approach.

## 6. Related work

We categorize the vast body of research related to the scope of our work into four groups. We first list the standardization activities, commercial solutions and important research efforts tackling the design of edge computing systems. Second, narrowing the scope, we collect the most relevant academic papers that describe work aiming at the scalable and reliable resource orchestration and/or service management of such systems, highlighting those that pay special attention to network-related requirements. Third, related to the latter, we give an overview on the literature of embedding and scheduling algorithms. Finally, prior art of the implementation aspects of integrating VIMs in the orchestration platforms is collected.

### 6.1. Standardization and commercial edge solutions

Many standardization bodies, academia and industries such as

Institute of Electrical and Electronics Engineers (IEEE) and European Telecommunications Standards Institute (ETSI) lead activities on edge networks to identify standardization opportunities and gaps. In the United States, these activities are managed by the National Science Foundation (NSF) Future Internet Architecture initiative and in Europe, they are lead under the European Union Framework programs H2020 and Horizon Europe.

ETSI is one of the key players in setting telecommunication standards; their purpose is to create a standardized and open environment for edge computing. ETSI coined the term Mobile Edge Computing (MEC) (Mobile-Edge Computing, 2015) in December 2014 with a white paper authored by Huawei, IBM, Intel, Nokia, NTT DOCOMO, and Vodafone, defining the aim to shift storage, processing, and control to the edge of the network, specifically to Radio Access Networks (RAN). In MEC, third-party Application Service Providers are offered cloud computing capabilities at the edge of the network with strong focus on mobile scenarios. However, nowadays MEC support is provided for both fixed and mobile networks. IEEE also started a 5G Working Group (IEEE 5G Initiative, 2017) and showcased their vision and goals in a white paper (IEEE 5G Technical Community, 2017).

In 2015 the Open Edge Computing project (Open Edge Computing) was founded with special emphasis on prototyping applications with edge computing. In November 2015 Open Fog Consortium (OpenFog) was created (ETSI and OpenFog Consortium) by ARM, Cisco, Dell, Intel, Microsoft and Princeton University in order to solve the challenges such as latency, bandwidth and communications of advanced concepts, such as Tactile Internet, IoT, Artificial Intelligence and Robotics. A group called IMT-2020 (SG13) was created by ITU Telecommunication Standardization Sector (ITU-T) to identify and study how 5G technologies will interact in future networks. 5G Americas presented a white paper (White Paper, 2016) that suggests the 5G requirements in terms of new protocols and architectures with emphasis on caching, mobility and latency. Next Generation Mobile Networks (NGMN) states the requirements and adoption of edge computing in their white paper (NGMN 5G White Paper, 2015). The EdgeX Foundry (The Linux Foundation Projects) project for IoT edge computing was initiated by Linux Foundation with the aim to develop an edge computing platform for IoT ecosystems. However, the approach is strongly focusing on industrial IoT devices and is limited to microservices. Eurotech presented another IoT edge computing platform called the Everyware Software Framework (Eurotech) which supports edge applications for IoT devices. The platform is developed on the modular Open Service Gateway initiative (OSGi).

In addition to standardization bodies, a lot of vendors are working on the hardware and software solutions of edge computing and future networking. In this regards, ADLINK technology (ADLINK Technology) is very active and provides hardware devices with fog computing and MEC features. A product named SETO-1000 (Extreme Outdoor Server) was launched by ADLINK which is also a part of the MEC architecture: SETO-1000 provides mini cloud like facility closer to the users in RAN. Moreover, ADLINK is active and has many contributions in standardized bodies such as OpenFog Consortium, Telecom Infra Project (TIP), the PCI Industrial Computer Manufacturers Group (PICMG), the PXI Systems Alliance (PXISA), Open Compute Project (OCP), ETSI MEC and Network function virtualization (NFV), and the Standardization Group for Embedded Technologies (SGET). Advantech (Advantech) also provides hardware solutions such as Packetarium XLc, which is a virtualized platform for edge computing deployments. Artesyn (Artesyn) designs hardware for next generation networks and has developed MaxCore platform for edge computing. The main focus of MaxCore is to enhance the platform performance in terms of latency and bandwidth for high dense traffic environments. Interdigital (Interdigital) is also working on edge computing, SDN and strongly focusing on the research and development of 5G. Qwilt (Qwilt) offers broadband fixed and wireless services. Qwilt provides solutions that extend Content Delivery Networks (CDNs) with the objective of reducing transport cost and making the content delivery more efficient. For that Open Caching software provides quick access to popular content without requesting any action from CDNs. Vasona Networks (Vasona Networks) provides solutions to optimize RAN performance and provides better QoE to mobile operators while using network resources efficiently. Vasona developed standard-based platforms for MEC that can cover more than a thousand cells and can be placed at aggregation points between RAN and core networks. For the cell level a product named SMART AIR is being developed that controls the traffic flows at real time. Another product called SmartVISION is provided by Vasona that offers real-time assistance to operators based on historical data and user activity in a cell, which can be used for planning and designing networks.

Several recent surveys (Shi et al., 2016; Mach and Becvar, 2017; Taleb et al., 2017a; Pan and McElhannon, 2018) summarize use cases, fundamental key enabling technologies and orchestration deployment options of edge computing. In parallel, standardization activities are ongoing at ETSI (Multi-access Edge Computing, 2019) and at the OpenFog consortium (OpenFog Reference Architecture for Fog Computing, 2017), both providing their respective MEC reference architecture. Various MEC schemes are also proposed from the academia, specifically designed e.g., for smart city scenarios (Taleb et al., 2017b), and IoT services (Villari et al., 2016).

### 6.2. Orchestration systems

There is plenty of NFV orchestration solutions proposed by researchers, and many products are available for this purpose. We depict a summary of the features the most prominent orchestration frameworks offer in Table 2: we highlight whether the selected orchestration systems provide the possibility of defining delay constraints for the applications they schedule, and whether they support multiple domains. These two features are the ones that distinguish our proposed systems from the existing frameworks. The one with the most hype around is Kubernetes (Kubernetes); it manages light-weight hypervisors (Linux Containers; Docker) typically for the deployment of micro-services into containers. For big data applications, Apache YARN (Apache Hadoop YARN), Mesos (Apache Mesos) and Marathon (Marathon) are the most commonly used technologies in the resource management layer of Hadoop, the *de facto* big data framework. These solutions schedule computational resources for applications with little awareness to network parameters, e.g., they estimate network bandwidth capacity based on the physical proximity of two servers. This estimation works well within one data center, though it fails in a multi-cloud environment. To remedy this shortcoming, placement solutions for MapReduce tasks deployed in a geographically distributed environment propose moving the input data (Ruiz-Alvarez and Humphrey, 2014; Cavallo et al., 2016; Heintz et al., 2016; Zhang et al., 2014). In contrast, our solutions also consider the networking capabilities of the underlying domains.

Besides the control and orchestration frameworks targeting solely IT resources, integrated architectures and experimental solutions address the joint control of compute and network resources (Network Functions Virtualisation, 2014; Open Source MANO; OPEN-O; OPNFV; CORD; Ciena Blue Planet MDSO; OpenBaton; Tacker; ONAP). Most open source control and orchestration frameworks are based on the ETSI NFV MANO specification (Network Functions Virtualisation, 2014) and thanks to

**Table 2**
Features of existing orchestration systems.

| Name | Orchestrated entity | Delay constraint | Multiple domains |
|---|---|---|---|
| Openstack | VM | – | – |
| Kubernetes | Pod | – | – |
| YARN | job | – | ✓ |
| Mesos | task | – | – |
| Marathon | container | – | – |
| OSM | VM | – | ✓ |

their modular architectural design, developers are able to replace each component to ensure collaboration with third party software. One of the most prominent implementations is OSM (Open Source MANO) that we use as baseline in our performance evaluation, described in Sec. 5.

While most of the solutions provide multi-VIM support, the inter-VIM orchestration feature is rudimentary in each of them. Therefore Guerzoni et al. (2017), Sun et al. (2018a) and Bhamare et al. (2017) studied the problem of SFC orchestration across multiple domains and/or for a multi-cloud setup, and proposed functional architectures for the end-to-end service management and orchestration plane, although with slightly different optimization goals in mind, e.g., minimizing inter-domain traffic and response time. In Sun et al. (2018b) the authors proposed an alternative to SFC-based service abstraction, and for their model, they designed an efficient placement method specifically for edge computing topologies. Zanzi et al. (2018) introduced the concept of MEC broker as an entity exposing administration and management capabilities while handling heterogeneous tenant privileges. Their orchestration solution optimally allocates requested resources in compliance with the tenants' service level agreements. In Sun et al. (2018a), the virtual links of the service requests describe only bandwidth demands, while in Bhamare et al. (2017) and Zanzi et al. (2018), only the maximum delays tolerated by the users are included in the service level agreements (SLAs). In contrast, our solutions provide network-aware orchestration over edge and cloud resources taking both delay and bandwidth requirements into account during the service deployment. In addition, the solutions presented in Sun et al. (2018a, 2018b) and Bhamare et al. (2017) do not consider migration as an extra step for better utilization, whereas the mapping process described in Zanzi et al. (2018) supports VNF migration but based only on delay information. On the contrary, our VIM extension (DARK) is able to migrate VNFs for better utilization while taking both bandwidth and delay characteristics into consideration.

An example for cross-layer service-specific orchestration is shown in He et al. (2017): the authors proposed an *integrated* framework that enables dynamic orchestration of networking, caching, and computing resources to improve the performance of applications for smart cities.

In contrast to the aforementioned solutions, our multi-layer orchestration system inherently supports different aspects of multi-domain and multi-provider scenarios, including techniques for information hiding and aggregation together with embedding methods resolving end-to-end constraints.

### 6.3. *Network embedding and task scheduling*

Virtual Network Embedding is the process that maps multiple graphs (representing the services composed by interconnected VNFs) (Németh et al., 2016) to a common physical infrastructure, represented by a resource graph. The VNE problem is known to be $\mathcal{NP}$-hard (Amaldi et al., 2016), and it has been addressed by a plethora of research initiatives (Amaldi et al., 2016; Chowdhury et al., 2012; Fuerst et al., 2013; Bari et al., 2015). Two different approaches emerged for solving the problem: *i*) exact solutions that find solution but these can be applied to limited scale problems only, *ii*) approximation-based algorithms that trade the optimal solution for better runtime. Fischer et al. (2013) summarizes many solutions for both. In the following we describe selected scheduling methods from prior art that are capable of orchestrating applications consisting of multiple components (e.g., SFC) in an online manner, i.e., one-by-one as application deployment requests arrive.

Placing service components into the infrastructure is tackled in Guo et al. (2018) as a mobile-edge computation offloading problem in ultra-dense IoT networks: the authors propose a two-tier game-theoretic greedy offloading scheme. Another approach is examined in Zhu and Huang (2017) where authors develop a cost-efficient placement method for mobile edge applications considering availability and confidentiality requirement by applying affinity- and antiaffinity rules. Another mathematical problem related to service and resource orchestration is task scheduling. Alameddine et al. (2019) propose an approach of jointly deciding on the task offloading (placing tasks to MEC servers) and scheduling (order of executing them) for IoT devices and applications. By decomposing the complex joint optimization problem, the authors achieved improvements in the runtime of scheduling decisions. In Hassan et al. (2015), researchers study the problem of task offloading from mobile device to fog nodes to minimize the response time of the application. They leverage pre-trained Multilayer Perceptron models to estimate the performance of the tasks, but without considering the available resources on the fog nodes. Similarly, authors of Skarlat et al. (2017) and Xiao and Krunz (2017) propose solutions for leveraging edge resources in a fog environment to provide better service response times. Although, in both papers the authors apply distributed and cooperative approaches to achieve efficient resource usage, they only consider directly connected neighbour nodes for workload propagation. In Bittencourt et al. (2017) researchers analyze the scheduling problem by focusing on how user mobility can influence application performance and how different scheduling policies can improve execution based on application characteristics. Authors of Zhang et al. (2019) propose a latency-aware edge resource orchestration platform over heterogeneous edge clouds. They support real-time responses to computation-intensive edge vision applications, completely relying on the Apache Storm framework.

Similarly to these solutions, by providing a network-aware extension to VIMs, DARK makes possible to manage both cloud and edge resources and orchestrate the required services over a geographically scattered infrastructure. On the other hand, differently from the related works mentioned above, DARK provides a novel migration mechanism to ensure the VNF migration within the managed infrastructure. By moving services, DARK is able to *i*) optimize the computation costs, *ii*) maximize the number of deployed services, and *iii*) adapt to end user/device mobility. Our other solution, i.e., MORCH, inherently supports several features required for multi-domain operations, such as automated resolution of end-to-end delay constraints, operation on abstract and limited/aggregated resource views, and a multi-layer, distributed backtracking mechanism, which are typically not addressed by available algorithms.

We highlight the capabilities of the discussed research results in Table 3 from the aspects that constitute the contribution of our proposed methods to the body of research: involvement of delay constraints in the scheduling decisions, the ability of managing a large, distributed infrastructure, and the feature of migrating deployed application components when it is deemed necessary.

### 6.4. *Virtual infrastructure adaptation*

In OpenStack the *nova-scheduler* component is responsible for managing computational resources on the hypervisor of each physical host. The placement strategy applied therein, called filter scheduler

**Table 3**
Features of scheduling algorithms in related work.

| Reference | Delay constraint | Multiple domains | VM/Pod/VNF migration |
|---|---|---|---|
| Guo et al. (2018) | – | ✓ | – |
| Zhu and Huang (2017) | – | – | – |
| Alameddine et al. (2019) | ✓ | – | – |
| Hassan et al. (2015) | – | ✓ | – |
| Skarlat et al. (2017) | ✓ | ✓ | – |
| Xiao and Krunz (2017) | ✓ | ✓ | – |
| Bittencourt et al. (2017) | ✓ | ✓ | – |
| Zhang et al. (2019) | – | – | – |

(OpenStack Nova Filter Scheduler), has several limitations. For one, the sequential processing of VM requests makes it impossible to define complex placement constraints that affect more than one instance. Furthermore, the filtering step currently does not consider any networking related metrics, which might be a shortcoming in a MEC infrastructure.

Various extensions to OpenStack were proposed (Scharf et al., 2015; Sahasrabudhe and Sonawani, 2015) for the support of network-aware placement of instances. These solutions take into account bandwidth constraints to and from nodes by keeping track of host-local network resource allocation. Authors of Lucrezia et al. (2015) introduced a network-aware scheduler that aimed at optimizing the VM placement from a networking perspective: they used OpenDayLight (OpenDayLight) to collect network topology information and to configure traffic steering with the goal of minimizing the bandwidth utilization of physical links. Haja et al. (2018) proposed a solution that alleviated the need for running OpenStack controllers in the lightweight edge, plus it took into account network aspects that are extremely important in a resource setup with remote fogs. In contrast to these solutions, DARK can take both the delay and bandwidth characteristics into consideration and in addition, it is able to migrate VNFs to achieve better utilization, which is typically not supported by available systems.

Using lightweight container-based virtualization techniques has also been investigated (Alam et al., 2018; Farris et al., 2017; Xiong et al., 2018) in order to design modular, scalable, distributed deployments for highly dynamic service deployment in MEC environments, e.g., by proactively exploiting service replication, or by ensuring a unified multi-tenant communication infrastructure between edge and cloud with fault tolerance and high availability.

## 7. Conclusion

The strong trend of virtualization has resulted in most online applications being containerized and run in VMs in the cloud instead of being deployed on bare metal in-house. The progress continues by the spread of fog and edge computing systems, which together with the advanced wireless radio technology of 5G and the plethora of smart devices and sensors of the Internet of Things, will provide the possibility of creating ultra time-critical online applications.

In this work, we presented two alternatives for the service orchestration in such distributed edge systems. We propose either to orchestrate the system in a completely flat architecture, or in a hierarchical recursive manner. The first option assumes that all the infrastructure islands are controlled by a single manager, i.e., OpenStack, so an extension is proposed to make it suitable for the distributed edge topology. The second option places an extra orchestration component next to each infrastructure manager, e.g., next to an edge node's Docker engine, and organizes them in a multi-layer topology. With both solutions our aim is to quickly and efficiently map incoming service deployment requests to physical resources.

We presented the design choices and implementation caveats in detail and we showed the performance of both solutions with simulated and emulated edge infrastructure. In order to evaluate the efficiency of the proposed solutions, we compared them to a production-quality orchestration system providing a baseline for a basic set of features.

## Credit author statement

**Balázs Sonkoly:** Conceptualization, Funding acquisition, Methodology, Project administration, Supervision, Writing - original draft. **Dávid Haja:** Investigation, Methodology, Software, Writing - original draft. **Balázs Németh:** Investigation, Methodology, Software, Writing - original draft. **Márk Szalay:** Investigation, Methodology, Software, Writing - original draft. **János Czentye:** Investigation, Methodology, Software, Writing - original draft. **Róbert Szabó:** Conceptualization, Methodology, Software. **Rehmat Ullah:** Writing - original draft, Writing - review & editing. **Byung-Seo Kim:** Funding acquisition, Writing - original draft, Writing - review & editing. **László Toka:** Conceptualization, Funding acquisition, Methodology, Supervision, Writing - original draft.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

Project no. 2018-2.1.17-TÉT-KR-2018-00012 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary. This research was also supported by the International Research & Development Program of the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT, Republic of Korea (No. NRF-2018K1A3A1A39086819).

## References

ADLINK Technology [Online]. https://www.adlinktech.com.
Advantech [Online]. https://www.advantech.com.
Alam, M., Rufino, J., Ferreira, J., Ahmed, S.H., Shah, N., Chen, Y., 2018. Orchestration of microservices for IoT using docker and edge computing. IEEE Commun. Mag. 56 (9), 118–123. https://doi.org/10.1109/MCOM.2018.1701233.
Alameddine, H.A., Sharafeddine, S., Sebbah, S., Ayoubi, S., Assi, C., 2019. Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing. IEEE J. Sel. Area. Commun. 37 (3), 668–682. https://doi.org/10.1109/JSAC.2019.2894306.
Amaldi, E., Coniglio, S., Koster, A.M., Tieves, M., 2016. On the computational complexity of the virtual network embedding problem. Electron. Notes Discrete Math. 52, 213–220. https://doi.org/10.1016/j.endm.2016.03.028.
Apache Hadoop YARN [Online]. https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.
Apache Mesos [Online]. https://mesos.apache.org.
Artesyn [Online]. https://www.artesyn.com.
Bari, M.F., Chowdhury, S.R., Ahmed, R., Boutaba, R., 2015. On orchestrating virtual network functions. In: 2015 11th International Conference on Network and Service Management (CNSM), pp. 50–56. https://doi.org/10.1109/CNSM.2015.7367338.
Bhamare, D., Samaka, M., Erbad, A., Jain, R., Gupta, L., Chan, H.A., 2017. Optimal virtual network function placement in multi-cloud service function chaining architecture. Comput. Commun. 102 (C), 1–16. https://doi.org/10.1016/j.comcom.2017.02.011.
Bittencourt, L.F., Diaz-Montes, J., Buyya, R., Rana, O.F., Parashar, M., 2017. Mobility-aware application scheduling in fog computing. IEEE Cloud Comput. 4 (2), 26–35. https://doi.org/10.1109/MCC.2017.27.
Cavallo, M., Di Modica, G., Polito, C., Tomarchio, O., 2016. H2F: a hierarchical Hadoop framework for big data processing in geo-distributed environments. In: 2016 IEEE/ACM 3rd International Conference on Big Data Computing Applications and Technologies (BDCAT), pp. 27–35. https://doi.org/10.1145/3006299.3006320.
Chowdhury, M., Rahman, M.R., Boutaba, R., 2012. ViNEYard: virtual network embedding algorithms with coordinated node and link mapping. IEEE/ACM Trans. Netw. 20 (1), 206–219. https://doi.org/10.1109/TNET.2011.2159308.
Ciena Blue Planet MDSO [Online]. http://www.blueplanet.com/products/multi-domain-service-orchestration.html.
CORD [Online]. https://www.opennetworking.org/cord/.
DARK [Online]. https://github.com/hsnlab/dark.
Docker: a better way to build apps [Online]. https://www.docker.com.
ETSI and OpenFog Consortium collaborate on fog and edge applications [Online]. https://www.etsi.org/newsroom/news/1216-2017-09-news-etsi-and-openfog-consortium-collaborate-on-fog-and-edge-applications.
EU H2020 5G Exchange project [Online]. https://github.com/5GExchange.
Eurotech: Edge Computing Platform [Online]. https://esf.eurotech.com/docs/edge-computing-platform.
Extreme Outdoor Server. ADLINK Technologies [Online]. https://adlinktech.com/Products/Server/Extreme_Outdoor_Server/SETO-1000.
Farris, I., Taleb, T., Iera, A., Flinck, H., 2017. Lightweight service replication for ultra-short latency applications in mobile edge networks. In: 2017 IEEE International Conference on Communications (ICC), pp. 1–6. https://doi.org/10.1109/ICC.2017.7996357.
Fischer, A., Botero, J.F., Beck, M.T., de Meer, H., Hesselbach, X., 2013. Virtual network embedding: a survey. IEEE Commun. Surv. Tutor. 15 (4), 1888–1906. https://doi.org/10.1109/SURV.2013.013013.00155.
Fuerst, C., Schmid, S., Feldmann, A., 2013. Virtual network embedding with collocation: benefits and limitations of pre-clustering. In: 2013 IEEE 2nd International Conference on Cloud Networking (CloudNet), pp. 91–98. https://doi.org/10.1109/CloudNet.2013.6710562.

Ger, B., Jocha, D., Szab, R., Czentye, J., Haja, D., Nmeth, B., Sonkoly, B., Szalay, M., Toka, L., Cano, C.J.B., Murillo, L.M.C., 2017. The orchestration in 5G exchange a multi-provider NFV framework for 5G services. In: 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 1–2. https://doi.org/10.1109/nfv-sdn.2017.8169865.

Guerzoni, R., Vaishnavi, I., Perez-Caparros, D., Galis, A., Tusa, F., Monti, P., Sganbelluri, A., Biczk, G., Sonkoly, B., Toka, L., Ramos, A., Melin, J., Dugeon, O., Cugini, F., Martini, B., Iovanna, P., Giuliani, G., Figueiredo, R., Miguel Contreras-Murillo, L., Szabo, R., 2017. Analysis of end-to-end multi-domain management and orchestration frameworks for software defined infrastructures: an architectural survey. Trans. Emerg. Telecomm. Technol. 28 (4), 1–19. https://doi.org/10.1002/ett.3103.

Guo, H., Liu, J., Zhang, J., Sun, W., Kato, N., 2018. Mobile-edge computation offloading for ultradense IoT networks. IEEE Internet Things J. 5 (6), 4977–4988. https://doi.org/10.1109/JIOT.2018.2838584.

Haja, D., Szab, M., Szalay, M., Nagy, ., Kern, A., Toka, L., Sonkoly, B., 2018. How to orchestrate a distributed OpenStack. In: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 1–2. https://doi.org/10.1109/INFCOMW.2018.8407014.

Hassan, M.A., Xiao, M., Wei, Q., Chen, S., 2015. Help your mobile applications with fog computing. In: 2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking - Workshops (SECON Workshops), pp. 1–6. https://doi.org/10.1109/SECONW.2015.7328146.

He, Y., Yu, F.R., Zhao, N., Leung, V.C.M., Yin, H., 2017. Software-defined networks with mobile edge computing and caching for smart cities: a big data deep reinforcement learning approach. IEEE Commun. Mag. 55 (12), 31–37. https://doi.org/10.1109/MCOM.2017.1700246.

Heintz, B., Chandra, A., Sitaraman, R.K., Weissman, J., 2016. End-to-End optimization for geo-distributed MapReduce. IEEE Trans. Cloud Comput. 4 (3), 293–306. https://doi.org/10.1109/TCC.2014.2355225.

Tech. rep. IEEE 5G and beyond Technology Roadmap White Paper, Oct. 2017. IEEE 5G Technical Community [Online]. https://futurenetworks.ieee.org/images/files/pdf/ieee-5g-roadmap-white-paper.pdf.

IEEE 5G Initiative, 2017. IEEE 5G Technology Roadmap [Online]. http://5g.ieee.org/roadmap.

Interdigital [Online]. http://www.interdigital.com.

Kubernetes: Production-Grade Container Orchestration [Online]. https://kubernetes.io.

Linux Containers [Online]. https://linuxcontainers.org.

Logical architecture of Openstack [Online]. https://docs.openstack.org/install-guide/get-started-logical-architecture.html.

Lucrezia, F., Marchetto, G., Risso, F., Vercellone, V., 2015. Introducing network-aware scheduling capabilities in OpenStack. In: Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft), pp. 1–5. https://doi.org/10.1109/NETSOFT.2015.7116155.

Mach, P., Becvar, Z., 2017. Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun. Surv. Tutor. 19 (3), 1628–1656. https://doi.org/10.1109/COMST.2017.2682318.

Marathon: a container orchestration platform for Mesos and DC/OS [Online]. https://mesosphere.github.io/marathon.

Mobile-Edge Computing (MEC); Service Scenarios, Nov. 2015. Tech. Rep. DGS/MEC-IEG004, ETSI, [Online]. https://www.etsi.org/deliver/etsi_gs/MEC-IEG/001_099/004/01.01.01_60/gs_MEC-IEG004v010101p.pdf.

Multi-access Edge Computing (MEC); Framework and Reference Architecture, Jan. 2019. Tech. Rep. RGS/MEC-0003v211Arch, ETSI, [Online]. https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/02.01.01_60/gs_MEC003v020101p.pdf.

Nmeth, B., Sonkoly, B., Rost, M., Schmid, S., 2016. Efficient service graph embedding: a practical approach. In: 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 19–25. https://doi.org/10.1109/NFV-SDN.2016.7919470.

Network Functions Virtualisation (NFV); Management and Orchestration, Dec. 2014. Tech. Rep. DGS/NFV-MAN001, ETSI, [Online]. https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf.

Tech. Rep. v1 NGMN 5G White Paper, Feb. 2015. NGMN Alliance [Online]. https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf.

ONAP [Online]. https://www.onap.org.

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

OPEN-O [Online]. https://www.open-o.org.

Open Source MANO [Online]. https://osm.etsi.org.

OPNFV [Online]. https://www.opnfv.org.

OpenBaton [Online]. http://openbaton.github.io.

OpenDayLight [Online]. https://www.opendaylight.org.

OpenFog Reference Architecture for Fog Computing, Feb. 2017. Tech. Rep. OPFRA001.020817, OpenFog Consortium, [Online]. https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf.

OpenStack Nova Filter Scheduler [Online]. https://docs.openstack.org/nova/latest/user/filter-scheduler.html.

OpenStack Service Function Chaining [Online]. https://docs.openstack.org/newton/networking-guide/config-sfc.html.

Pan, J., McElhannon, J., 2018. Future edge cloud and edge computing for Internet of Things applications. IEEE Internet Things J. 5 (1), 439–449. https://doi.org/10.1109/JIOT.2017.2767608.

Qwilt [Online]. https://www.qwilt.com.

Ren, J., Zhang, D., He, S., Zhang, Y., Li, T., 2019. A survey on end-edge-cloud orchestrated network computing paradigms: transparent computing, mobile edge computing, fog computing, and cloudlet. ACM Comput. Surv. 52 (6) https://doi.org/10.1145/3362031.

Rost, M., Schmid, S., 2020. On the hardness and inapproximability of virtual network embeddings. IEEE/ACM Trans. Netw. 28 (2), 791–803.

Ruiz-Alvarez, A., Humphrey, M., 2014. Toward optimal resource provisioning for cloud MapReduce and hybrid cloud applications. In: Proceedings of the 2014 IEEE/ACM International Symposium on Big Data Computing (BDC), pp. 74–82. https://doi.org/10.1109/BDC.2014.14.

Sahasrabudhe, S., Sonawani, S.S., 2015. Improved filter-weight algorithm for utilization-aware resource scheduling in OpenStack. In: 2015 International Conference on Information Processing (ICIP), pp. 43–47. https://doi.org/10.1109/INFOP.2015.7489348.

Scharf, M., Stein, M., Voith, T., Hilt, V., 2015. Network-aware instance scheduling in OpenStack. In: 2015 24th International Conference on Computer Communication and Networks (ICCCN), pp. 1–6. https://doi.org/10.1109/ICCCN.2015.7288436.

SG13: future networks, with focus on IMT-2020, cloud computing and trusted network infrastructures [Online]. https://www.itu.int/en/ITU-T/studygroups/2017-2020/13.

Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L., 2016. Edge computing: vision and challenges. IEEE Internet Things J. 3 (5), 637–646. https://doi.org/10.1109/JIOT.2016.2579198.

Skarlat, O., Nardelli, M., Schulte, S., Dustdar, S., 2017. Towards QoS-aware fog service placement. In: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), pp. 89–96. https://doi.org/10.1109/ICFEC.2017.12.

Sonkoly, B., Szab, R., Jocha, D., Czentye, J., Kind, M., Westphal, F.-J., 2015. UNIFYing cloud and carrier network resources: an architectural view. In: 2015 IEEE Global Communications Conference (GLOBECOM), pp. 1–7. https://doi.org/10.1109/GLOCOM.2015.7417869.

Sonkoly, B., Szab, M., Nmeth, B., Majdn, A., Pongrcz, G., Toka, L., 2018. FERO: fast and efficient resource orchestrator for a data plane built on docker and DPDK. In: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, pp. 243–251. https://doi.org/10.1109/INFOCOM.2018.8485953.

Sun, G., Li, Y., Liao, D., Chang, V., 2018a. Service function chain orchestration across multiple domains: a full mesh aggregation approach. IEEE Trans. Netw. Serv. Manag. 15 (3), 1175–1191. https://doi.org/10.1109/TNSM.2018.2861717.

Sun, G., Li, Y., Li, Y., Liao, D., Chang, V., 2018b. Low-latency orchestration for workflow-oriented service function chain in edge computing. Future Generat. Comput. Syst. 85, 116–128. https://doi.org/10.1016/j.future.2018.03.018.

Szalay, M., Haja, D., Dka, J., Sonkoly, B., Toka, L., 2019. Demo abstract: turning OpenStack into a fog orchestrator. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 947–948. https://doi.org/10.1109/INFCOMW.2019.8845035.

Tacker [Online]. https://wiki.openstack.org/wiki/Tacker.

Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., Sabella, D., 2017a. On multi-access edge computing: a survey of the emerging 5G network edge cloud architecture and orchestration. IEEE Commun. Surv. Tutor. 19 (3), 1657–1681. https://doi.org/10.1109/COMST.2017.2705720.

Taleb, T., Dutta, S., Ksentini, A., Iqbal, M., Flinck, H., 2017b. Mobile edge computing potential in making cities smarter. IEEE Commun. Mag. 55 (3), 38–43. https://doi.org/10.1109/MCOM.2017.1600249CM.

The Linux Foundation projects: EdgeX Foundry [Online]. https://www.edgexfoundry.org.

UNIFY virtualizer library [Online]. https://github.com/5GExchange/virtualizer.

VMTP [Online]. http://vmtp.readthedocs.io.

Vaishnavi, I., Czentye, J., Gharbaoui, M., Giuliani, G., Haja, D., Harmatos, J., Jocha, D., Kim, J., Martini, B., MeMn, J., Monti, P., Nmeth, B., Poe, W.Y., Ramos, A., Sgambelluri, A., Sonkoly, B., Toka, L., Tusa, F., Bernardos, C.J., Szab, R., 2018. Realizing services and slices across multiple operator domains. In: NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, pp. 1–7. https://doi.org/10.1109/NOMS.2018.8406168.

Vasona Networks [Online]. https://www.vasonanetworks.com.

Villari, M., Fazio, M., Dustdar, S., Rana, O., Ranjan, R., 2016. Osmotic computing: a new paradigm for edge/cloud integration. IEEE Cloud Comput. 3 (6), 76–83. https://doi.org/10.1109/MCC.2016.124.

Tech. Rep. v2, ETSI White Paper: Network Functions Virtualisation (NFV), Oct. 2013 [Online]. http://portal.etsi.org/nfv/nfv_white_paper2.pdf.

Tech. rep. White Paper: Understanding Information Centric Networking and Mobile Edge Computing, Dec. 2016. 5G Americas [Online]. https://www.5gamericas.org/wp-content/uploads/2019/07/Understanding_Information_Centric_Networking_and_Mobile_Edge_Computing.pdf.

Xiao, Y., Krunz, M., 2017. QoE and power efficiency tradeoff for fog computing networks with fog node cooperation. In: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, pp. 1–9. https://doi.org/10.1109/INFOCOM.2017.8057196.

Xiong, Y., Sun, Y., Xing, L., Huang, Y., 2018. Extend cloud to edge with KubeEdge. In: 2018 IEEE/ACM Symposium on Edge Computing (SEC), pp. 373–377. https://doi.org/10.1109/SEC.2018.00048.

Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., Jue, J.P., 2019. All one needs to know about fog computing and related edge computing paradigms: a complete survey. J. Syst. Architect. 98, 289–330. https://doi.org/10.1016/j.sysarc.2019.02.009.

Zanzi, L., Giust, F., Sciancalepore, V., 2018. M2EC: a multi-tenant resource orchestration in multi-access edge computing systems. In: 2018 IEEE Wireless Communications and Networking Conference (WCNC), pp. 1–6. https://doi.org/10.1109/WCNC.2018.8377292.

Zhang, Q., Liu, L., Lee, K., Zhou, Y., Singh, A., Mandagere, N., Gopisetty, S., Alatorre, G., 2014. Improving Hadoop service provisioning in a geographically distributed cloud.

In: 2014 IEEE 7th International Conference on Cloud Computing (CLOUD), pp. 432–439. https://doi.org/10.1109/CLOUD.2014.65.

Zhang, W., Li, S., Liu, L., Jia, Z., Zhang, Y., Raychaudhuri, D., 2019. Hetero-edge: orchestration of real-time vision applications on heterogeneous edge clouds. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, pp. 1270–1278. https://doi.org/10.1109/INFOCOM.2019.8737478.

Zhu, H., Huang, C., 2017. Availability-aware mobile edge application placement in 5G networks. In: GLOBECOM 2017 - 2017 IEEE Global Communications Conference, pp. 1–6.

**Balázs Sonkoly** is an associate professor at Budapest University of Technology and Economics (BME) and he is the head of MTA-BME Network Softwarization Research Group. He received his Ph.D. (2010) and M.Sc. (2002) degrees in Computer Science from BME. He has participated in several EU projects (FP7 OpenLab, FP7 UNIFY, H2020 5G Exchange) and national projects. He was the demo co-chair of ACM SIGCOMM 2018, EWSDN'15,'14, IEEE HPSR'15. His current research activity focuses on cloud / edge / fog computing, NFV, SDN, and 5G.

**Dávid Haja** is a Ph.D. student at Budapest University of Technology and Economics. He is a member of the High Speed Networks Laboratory (http://hsnlab.hu) at the Department of Telecommunications and Media Informatics. His main research interests include Edge Computing, Software-Defined Networking (SDN), Network Function Virtualization (NFV) and Resource Orchestration.

**Balázs Németh** is an Industrial Ph.D. student at Budapest University of Technology and Economics (BME) in cooperation with Ericsson Research. He obtained his M.Sc. degree at BME as a Computer Science Engineer in info-communication specialization (2016). He has been working on orchestration algorithms for the H2020 5G-PPP 5G Exchange (5GEx) project. Currently, he is pursuing his Ph.D. degree in network softwarization with special focus on orchestration algorithms and next generation network models.

**Márk Szalay** is a Ph.D. student at Budapest University of Technology and Economics. He is a member of the High Speed Networks Laboratory (http://hsnlab.hu) at the Department of Telecommunications and Media Informatics. His main research interests include hardware (router / switch / NIC) design, network programming, software-defined networking and network function virtualization.

**János Czentye** is currently in his third year of his Ph.D. studies at Budapest University of Technology and Economics (BME). He completed an M.Sc. (2014) in the topic of Networks and Services with highest honours. He worked at High Speed Networks Laboratory (HSNLab) contributing in research projects (FP7 UNIFY, H2020 5GEx) and gained wide knowledge about SDN/NFV, microservice and cloud technologies. His current Ph.D. research focuses on cloud native service modeling and provisioning.

**Róbert Szabó** is a principal researcher at Research Area Cloud Systems and Platform, Ericsson Research. Since joining Ericsson in 2013, he was the technical coordinator of the EU-FP7 Unifying Cloud and Carrier Networks (UNIFY) integrated project (2013–2016) and he was the project coordinator of the H2020 5G-PPP 5G Exchange (5GEx) innovation action (2015–2018). Recently, he is working with distributed / edge cloud, zero touch automation, and network function virtualization. He worked as an associate professor at Budapest University of Technology and Economics (BME), where he was the deputy head of the Dept. of Telecommunications and Media Informatics between 2008 and 2010. He holds a Ph.D. and M. Sc. in E.E., and an MBA from BME.

**Rehmat Ullah** is currently working as an Assistant Professor with the Department of Computer Engineering at Gachon University, Global Campus, South Korea. He received his B.S. and M.S. degrees in computer science (major in wireless communications and networks) from COMSATS University Islamabad, Pakistan, in 2013 and 2016, respectively and the Ph.D. degree in electronics and computer engineering from Hongik University, South Korea, in February 2020. His research focuses on the broader area of future Internet and network systems, particularly the development of architectures, algorithms, and protocols for emerging paradigms, such as information centric networking/named data networking (ICN/NDN), Internet of Things (IoT), cloud/edge/fog computing for IoT, 5G and beyond.

**Byung-Seo Kim** received his B.S. degree in Electrical Engineering from In-Ha University, In-Chon, Korea in 1998 and his M.S. and Ph.D. degrees in Electrical and Computer Engineering from the University of Florida in 2001 and 2004, respectively. He is currently Professor in Dept. of Software and Communications Eng., Hongik University, Korea. He is IEEE Senior Member and Associative Editor of IEEE Access. His research interests include the design and development of efficient wireless/wired networks including link-adaptable/cross-layer-based protocols, multi-protocol structures, wireless CCNs/NDNs, Mobile Edge Computing, physical layer design for broadband PLC, and resource allocation algorithms for wireless networks.

**László Toka** is assistant professor at Budapest University of Technology and Economics, vice-head of HSNLab (http://hsnlab.hu), and member of both the MTA-BME Network Softwarization and the MTA-BME Information Systems Research Groups. He obtained his Ph.D. degree from Telecom ParisTech in 2011, he worked at Ericsson Research between 2011 and 2014, then he joined the academia with research focus on software-defined networking, cloud computing and artificial intelligence.