5G Applications From Vision to Reality: Multi-Operator Orchestration

Balázs Sonkoly, *Associate Member, IEEE*, Róbert Szabó, Balázs Németh, *Graduate Student Member, IEEE*, János Czentye, *Graduate Student Member, IEEE*, Dávid Haja, *Graduate Student Member, IEEE*, Márk Szalay, *Student Member, IEEE*, János Dóka, *Graduate Student Member, IEEE*, Balázs P. Gerő, *Associate Member, IEEE*, Dávid Jocha, *Member, IEEE*, and László Toka, *Member, IEEE*

***Abstract*— Envisioned 5G applications and services, such as Tactile Internet, Industry 4.0 use-cases, remote control of drone swarms, pose serious challenges to the underlying networks and cloud platforms. On the one hand, evolved cloud infrastructures provide the IT basis for future applications. On the other hand, networking is in the middle of a momentous revolution and important changes are mainly driven by Network Function Virtualization (NFV) and Software Defined Networking (SDN). A diverse set of cloud and network resources, controlled by different technologies and owned by cooperating or competing providers, should be coordinated and orchestrated in a novel way in order to enable future applications and fulfill application level requirements. In this paper, we propose a novel cross domain orchestration system which provides wholesale XaaS (Anything as a Service) services over multiple administrative and technology domains. Our goal is threefold. First, we design a novel orchestration system exploiting a powerful information model and propose a versatile embedding algorithm with advanced capabilities as a key enabler. The main features of the architecture include i) efficient and multi-purpose service embedding algorithms which can be implemented based on graph models, ii) inherent multi-domain support, iii) programmable aggregation of different resources, iv) information hiding together with flexible delegation of certain requirements enabling multi-operator use-cases, and v) support for legacy technologies. Second, we present our proof-of-concept prototype implementing the proposed system. Third, we establish a dedicated test environment spanning across multiple European sites encompassing sandbox environments from both operators and the academia in order to evaluate the operation of the system. Dedicated experiments confirm the feasibility and good scalability of the whole framework.**

***Index Terms*— NFV, SDN, resource orchestration, 5G.**

I. Introduction

FUTURE network services and 5G applications, such as Tactile Internet, remote surgery, coordination and control of drone swarms, or Industry 4.0 use-cases, pose serious challenges to the underlying networks and clouds. On the one hand, evolved cloud infrastructures and platforms, which are evident results of last years' research efforts, provide the IT basis for future applications. On the other hand, networking is in the middle of a momentous revolution and important changes are mainly driven by Network Function Virtualization (NFV) and Software Defined Networking (SDN). The success of future 5G systems and applications highly depends on the novel methods addressing the integration and joint virtualization of cloud and network resources. Moreover, a diverse set of resources, i) controlled by different technologies, ii) owned and managed by cooperating or competing providers, should be coordinated and orchestrated in a novel system. In order to enable emerging network services in carrier scale with strict QoS requirements, including end-to-end latency bounds or controlled dependability/reliability, we need a novel way to jointly control cloud and network resources instead of traditional approaches following separate control. In addition, the technological basis has to be elaborated which enables future business models and cooperation among the stakeholders of the ecosystem. Relevant research efforts have been recently focused on these technical challenges by the industry and the academia. The novel solutions and technical background enable a novel way of provisioning virtual services, virtual resources, and in general, software products, over networked systems. In the 5G ecosystem, operators have several options for providing *services* starting from simpler NFVIaaS (NFV Infrastructure as a Service) and NaaS (Network as a Service) ending with fully fledged VNFaaS (Virtual Network Function as a Service) or any types of SaaS (Software as a Service).

In this paper, we propose a novel cross domain orchestration system which provides wholesale XaaS (Anything as a Service) services over multiple administrative and technology domains. The key enablers of the system are a *novel embedding engine* supporting several constraints and multi-layer operation, a *resource control interface* and an *information model* together with the corresponding workflows, realizing the joint virtualization and control of cloud and network resources in a flexible and programmable way. The proposed and implemented system supports automated operations over multi-layer orchestration hierarchies created on demand and

Manuscript received December 15, 2018; revised February 2, 2020; accepted March 2, 2020. Date of publication June 12, 2020; date of current version June 29, 2020. This work was supported in part by the European Commission through the H2020-ICT-2014 project 5GEx under Grant 671636. *(Corresponding author: Balázs Sonkoly.)*

Balázs Sonkoly, Balázs Németh, János Czentye, Dávid Haja, Márk Szalay, János Dóka, and László Toka are with MTA-BME Network Softwarization Research Group, Budapest University of Technology and Economics, H-1111 Budapest, Hungary (e-mail: sonkoly@tmit.bme.hu).

Róbert Szabó, Balázs P. Gerő, and Dávid Jocha are with Ericsson Research, H-1117 Budapest, Hungary (e-mail: robert.szabo@ericsson.com).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2020.2999684

spanning across multiple, possibly administratively different, infrastructure domains.

Our goal is threefold. First, we provide the design of our novel *orchestration system for 5G* together with our versatile embedding algorithm enabling recursive orchestration hierarchies and advanced service requirements. The key "glue" between the components of the architecture is a powerful information model. The main features of the architecture include i) efficient service embedding algorithms which can be implemented based on the graph models, ii) inherent multi-domain support, iii) programmable aggregation of different resources, iv) information hiding together with flexible delegation of certain requirements enabling multi-operator use-cases, and v) support for legacy technologies. Second, we present our proof-of-concept prototype implementing the proposed system. Third, we establish a dedicated test environment spanning across multiple European sites encompassing sandbox environments both from operators and the academia, and then we evaluate the operation of the system. Control- and data plane experiments confirm the feasibility and good scalability of the whole framework.

The rest of the paper is organized as follows. Sec. II provides a summary on the related work. In Sec. II-B, an Industry 4.0 service is described in detail as an illustration of future services and requirements. Sec. III is devoted to the details of the system design including our information model and the resource control API. Sec. IV focuses on the embedding algorithm and its main features. In Sec. V, we present our proof-of-concept prototype implementing the relevant parts of our orchestration system. Sec. VI gives a detailed evaluation of the control plane operation and in addition, presents real experiments conducted across an European multi-domain sandbox environment including data planes with different technologies. And finally, Sec. VII draws the conclusions.

II. Background and Related Work

A. Virtualized Services, Federated Operators

Throughout the paper we utilize the notions of NFVIaaS, VNFaaS and Slice as a Service (SlaaS). The SlaaS provider offers supporting services, such as configuration and management of the elements of the service, and it manages the service in a holistic end-to-end manner, i.e., regardless of whose VNFs are deployed where, the end-to-end chain fulfills the requirements on the network and the service access point(s). Thus, the provisioning of end-to-end services requires the combination of services from all three aforementioned categories.

Ordinary network slicing allows a network operator to provide dedicated virtual networks with functionality specific to the service or to the customer over a common network infrastructure [28], [32]. Using resources such as processors and storage, network slicing also permits the creation of slices devoted to logical, self-contained, partitioned network functions [34]. *Federated network slicing* is designed to enable provisioning of network slices globally, making sure that customers do not need individual agreements with different operators for a global service experience [20]: early in 2017, a proof of concept demonstration of federated network slicing was shown at Mobile World Congress (MWC) by Ericsson, Deutsche Telekom and SK Telecom. Other vendors and operators, e.g., Huawei and British Telecom, are also intensively working on federated network slicing.

A network slice is inherently crafted for a specific single service and a single customer/tenant. Making a step further from the federated network slicing concept, the Slice as a Service model we propose is a composite, service-agnostic, wholesale offering of IaaS, VNFaaS and related enabling services.

B. An Illustrative Example of Future Services

This section is devoted to present a potential future use-case that illustrates emerging requirements, which should be fulfilled by 5G applications and also showcasing the challenges that orchestration systems and providers will face. Fig. 1 shows an envisioned Industry 4.0 robotics service (platform) and the relation between the stakeholders involved in provisioning and implementation of the service. Industry 4.0 is a name for the current trend of automation and data exchange in manufacturing technologies [5]. For the sake of clarity, the control and management planes are shown separately from the data and service planes in the figure. On the left hand side, the structure of the orchestration-related components and the requests from the customers are presented, while the right part of the figure illustrates how the orchestration system maps the service function chains (SFCs) to the underlying data plane infrastructure.

Multiple 5G operators with different roles, capabilities, hardware/software resources and service portfolios are cooperating in creation, deployment and provisioning of the robot service. This federation of providers makes 5G a global platform and enables service provisioning over the aggregated resource set of the whole federation.

By these means, the geographic footprint of an individual operator is extended to a "global coverage", which could be an important incentive for operators to participate. In our example, five different 5G operators/providers are involved, namely the Factory Provider, Operators 1-3, and the XaaS Provider.

Factory Provider is an NFVIaaS provider at the factory's premises with compute resources (small datacenter onsite), network infrastructure (SDN and industry WiFi in the example) and resource orchestration capabilities. *Operators 1-3* are 5G operators with geographically distributed footprints (e.g. from different countries), which are connected in the control plane and in the data plane as well if they own physical infrastructure. In our example, Op. 3 has direct connection (both control- and data plane) to the Factory Provider, however, Op. 1 has only indirect connections via Op. 2 or via Op. 2 and Op. 3. In the illustrated scenario, Op. 2 is a transport provider offering only network connectivity service (NaaS), while Op. 1 and Op. 3 are (enterprise) customer-facing providers with VNFaaS solutions. They provide fully fledged virtual services, consisting of own or 3rd party VNF software packages, service specific configuration and operation tools and service agnostic life-cycle management components, over a virtual 5G infrastructure including own and leased resources.

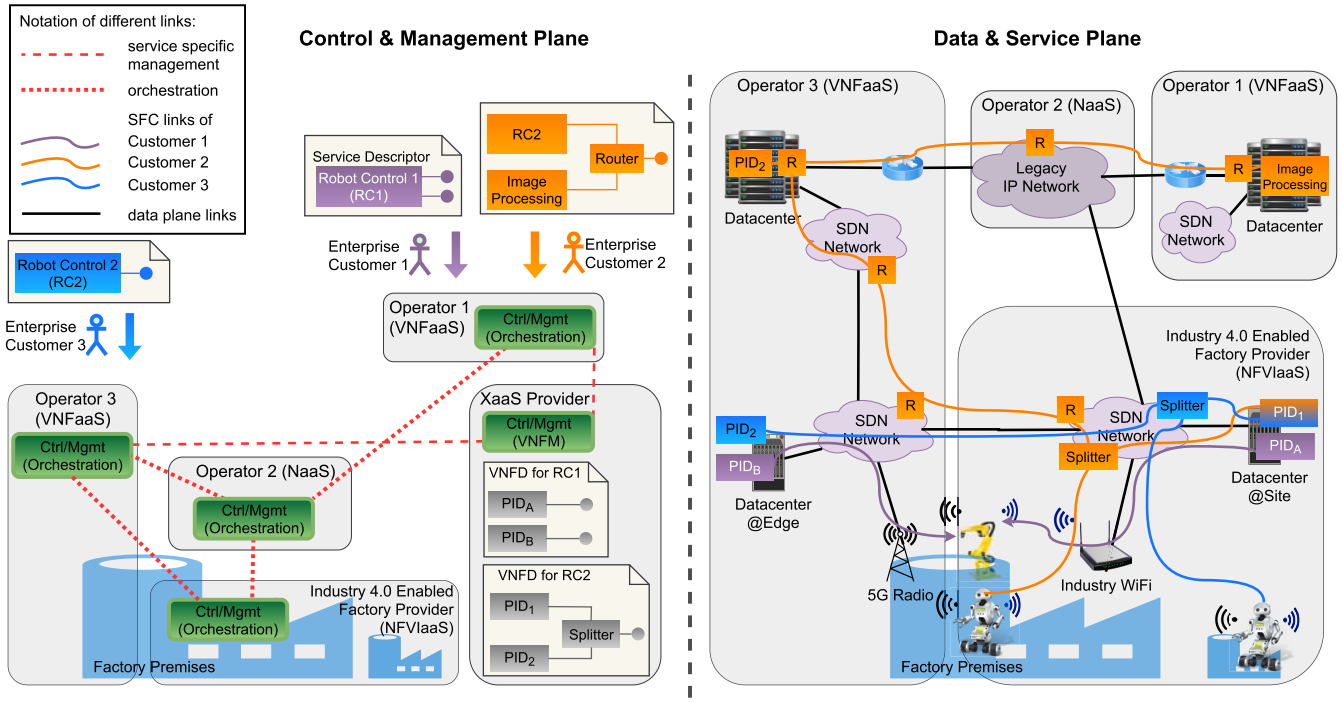


Fig. 1. An Industry 4.0 robot control service.

In order to construct the service, the 3rd party *XaaS Provider* is invoked who is aware of the service specific logic, service level requirements and decomposition rules. In addition, operators without data plane infrastructure could also participate in the orchestration workflow (not shown in the figure).

In our example, the XaaS Provider runs a dedicated VNF Manager (VNFM) module providing two different Robot Control VNFs (RC1, RC2) with different purpose, characteristics and requirements. On the one hand, RC1 can control a robotic arm in the factory, which requires ultra high reliability and strict latency bounds. This VNF is described by a dedicated VNF Descriptor (VNFD for RC1) available at the XaaS Provider. On the other hand, RC2 implements a resilient PID (proportional-integral-derivative) controller for a robot balancing vertically on two wheels (defined by VNFD for RC2). Customers can request these individual services directly from their own operators or they can construct compound services, formulated as novel service descriptors, in order to extend the basic features. For example, Customer 2 combines the RC2 VNF, managed by the 3rd party XaaS Provider, an unmanaged image processing VNF brought by the customer, and a distributed router VNF managed by internal VNFMs of the operators (not shown in the figure), while Customer 1 and Customer 3 request the default RC1 and RC2 services from their own operators, respectively.

A key and challenging task of the multi-domain and multi-operator orchestration system is to deploy the requested services on the available resources in an efficient way while meeting the service level requirements. For example, the envisioned robot service poses dependability and latency related requirements. Specifically, the ultra high reliability of RC1 should be provided by two VNFs running on physically different servers and connected via disjoint paths (*node/link anti-affinity constraints*), while the strict *latency bounds* should be resolved by VNFs placed close enough to the device. The realization of the *resilient* PID controller of RC2 needs a PID close to the robot and a backup PID without strict delay constraint. (We assume stateless PID VNFs that require all state information to be exchanged in the robot control messages.)

These types of constraints, including *end-to-end latency* and *bandwidth* requirements, *affinity* and *anti-affinity* rules, should be resolved in an automated way by the system, which requires the cooperation of different orchestrator modules. Moreover, in order to provision the service for Customer 2, cooperating providers should be able to connect RC2 and the image processing VNF via different technological domains (e.g., SDN and legacy IP), while meeting e.g., the latency and bandwidth constraints. In our example, the virtual distributed router should be *decomposed* and implemented by different domains accordingly, depending on the current placement of other components. Furthermore, XaaS provider should be able to *share* different *VNFs* among customers if policies enable that. In Fig. 1, PID1 (the one closer to the robot) is shared between Customers 2 and 3.

Today, it is challenging to create a service on-the-fly with given requirements spanning across multiple administrative domains. We argue that a situational orchestration hierarchy has to be constructed on-demand based on a negotiation process among 5G operators. As a result, an automated multi-layer orchestration process can be executed involving the cooperating operators. Resolving constraints, such as end-to-end latency/bandwidth constraints, VNF/link affinity and anti-affinity criteria, require novel workflows and inter-operation

between orchestrator components. This paper proposes an orchestration system supporting these novel features.

C. State-of-the-Art of Service and Resource Orchestration

There is plenty of orchestration solutions available in academia, and in the commercial offer. In this section, we categorize the vast body of work in this field along their field of application: whether the solution provides orchestrating features for cloud or for telco systems.

1) Cloud Computing and Micro Services: There are several fully-fledged solutions for IT resource and VM management: cloud platforms [1], [14], and cloud orchestrators [3] for both private and public clouds. The current hype builds upon light-weight hypervisors [4], [7], and cloud native orchestration solutions typically manage micro-services deployed in containers [6]. On the networking side, SDN platforms [11], [19] are available for controlling network resources and programming the low-level behavior of the forwarding and processing elements.

2) Telco Solutions for Network Function Virtualization: Besides control and orchestration frameworks targeting solely IT or solely network resources, a number of integrated architectures and experimental solutions address the joint control of these resources [2], [12], [13], [15]–[18], [21]. Nowadays many of the popular open source control and orchestration frameworks projects are based on the ETSI NFV MANO [21] specification and due to their modular architecture design, the developers are able to replace each component to ensure collaboration with third party software, which carries the benefits of fast development and enrichment of their features. While most of the solutions provide multi-VIM (Virtual Infrastructure Manager) support, the inter-VIM orchestration feature is still somewhat rudimentary in each of them.

Orchestrating over multiple domains is needed to deploy geographically scattered network services, however, at this time, these frameworks support only single operator mode, that is, each VIM island belongs to the same operator. Nevertheless, the service embedding problem is hard even if no inter-operator relations exacerbate the situation. This so-called Virtual Network Embedding problem has been addressed by a plethora of research initiatives [24]–[27], [29].

The Open Network Automation Platform (ONAP) [10] aims to offer policy-driven orchestration for different technology domains. Primarily, ONAP's orchestration engine lacks network resource (and topology) awareness, meaning it cannot take into account any requirements on the logical connections between the service components, such as latency and path reliability constraints. Furthermore, due to the lack of support for hierarchical orchestration in ONAP, it does not enable the federation of providers, i.e., it does not let service-focused 3rd party providers enter the 5G ecosystem. Contrary to ONAP, our solution can solve these challenges.

Our goal is to provide a multi-operator collaboration framework supporting special constraints on reliability and latency, which are resolved either directly, or delegated via an orchestration hierarchy. In ETSI, the Management and Orchestration Framework has been extended to support multiple administrative domains for NFVIaaS and NS across different administrative domains [23].

Metro Ethernet Forum's Third Network concept [8] envisions agile and assured global connectivity services that will be orchestrated end-to-end across all network domains, which may be owned and orchestrated by different network operators. Large operators and solution providers (with Cisco and Comarch on board) formed the *ngena* [9] alliance with the common goal of realizing network services in shared partner networks around the globe, however, technical details are not disclosed to the public. Nevertheless, to the best of our knowledge, currently there is no other orchestration system that would feature end-to-end latency and reliability support (later presented in Sec. V-A) across multiple administrative domains.

III. System Design

In this section, we summarize our main design goals and followed principles.

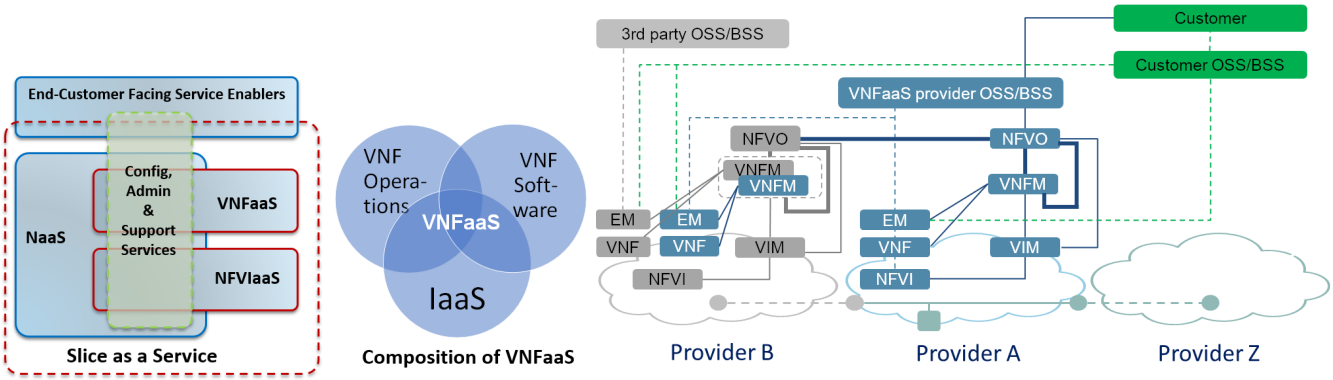
A. Design Principles

The main driving force of building a cross-domain orchestration system is the need for the novel end-user services that geographically span over multiple operators' turf, and are provisioned by multiple actors ranging from infrastructure providers to application providers.

Envisioned future applications can have special, even extreme requirements as we described in Sec. II-B. For example, critical machine-type communications use-cases require low latency, high reliability and continuous availability for a *wide geographic reach*.

The customer-facing provider contracted with the end-user may not have its own compute and network infrastructure anywhere the end-user might be interested to consume the service. Therefore, the option of utilizing resources of partner providers is essential, and for the envisioned 5G services, the federated network slicing is a must, as it is shown by the example given in Sec. II-B. In addition to buying and selling virtualized resources, there is a growing interest in *"outsourcing" the development of software components* as Virtual Network Functions (VNFs). The cross-domain orchestration system should support service composition, building on 3rd party application providers' VNFs as components. Furthermore, it should also support automated on-boarding of *3rd Party Product (3PP) VNFs* in order to enrich the VNF portfolio of given providers.

In this ecosystem, a customer-facing provider can buy virtual resources and VNFs from other partners so it can offer orchestrated services for their customers over the complete footprint of the federation. As all participating providers are separate business entities, they want sufficient control on what to expose from their resources, topology and services and to whom. In order to enable the federated orchestration and coordination, corresponding APIs should be defined and implemented. Besides the necessary technical alignment, comprehensive business agreements need to be made based on exchanging resource and service catalogs, sharing high level



(a) IaaS, VNFaaS and Slice as a Service

(b) Proposed architecture: extending ETSI MANO

Fig. 2. Services and our architecture proposal.

topology information for the expectable quality of service, automated price negotiation, SLA creation and enforcement, and finally charging plans. Users of this platform take interchangable provider and customer roles depending on who offers the end-user service and who provides the necessary building blocks for it.

The main concepts on various service options and relations among them are depicted in Fig. 2a. Following ETSI's terminology [22], NFVIaaS corresponds to Infrastructure as a Service, capable of hosting VNFs, while NaaS corresponds to Networking as a Service. Any value-added network service is provided as VNF. Central to our concept is the VNF as a Service (VNFaaS) definition. VNFaaS is a managed service, which comprises i) an infrastructure where the VNF can be hosted; ii) the VNF software and iii) operations including virtualization lifecycle management, necessary Element Management (EM) and Operations Support Systems / Business Support Systems (OSS/BSS). We believe that any of the VNFaaS components may be provided by different business actors, in this case the customer-facing provider must collect all the components and coordinate service provisioning. Note that it is also possible that any of the VNFaaS components are located at the custumer. In this context Slice as a Service (SlaaS) is defined as interconnection of XaaS components corresponding to NaaS and VNFaaS, respectively with their service configurations and supportive services.

The proposed orchestration system heavily builds on the concept of separating service orchestration and resource orchestration, because cross-domain resource orchestration interfaces are always required to provide cross-domain services as opposed to service orchestration interfaces, which may remain provider-internal. Our main goal is to provide the technological building blocks of the envisioned resource orchestration platform, which can have significant role in future 5G systems.

B. Proposed Architecture: Key Components and Workflows

Our architecture proposal shown in Fig. 2b extends ETSI's NFV MANO (Management and Orchestration) framework [21] with additional OSS/BBS functionality to enable orchestration across multiple administrative domains. Our architecture inherently supports the functional split between the service related OSS functionality of the VNFaaS provider, the VNFaaS customer and the owner of the VNF software, i.e., the developer/vendor. We argue that a VNF software package consists of three main parts: i) software components implementing the service (VNF), ii) software components implementing the configuration management interface of the first element, referred to as EM, iii) and software component operating the previous two ones, i.e., it creates/starts/stops them as needed, referred to as VNFM or lifecycle management (LCM). The latter two component groups represent OSS/BSS functionality. By realizing this split different VNFaaS ownership scenarios can be realized where the control and ownership of VNFs, EMs and VNFMs are shared among participating stakeholders (software vendors, VNFaaS providers, partner providers, customers) in any combination. In Fig. 2b, Provider A is the customer-facing VNFaaS provider consuming resources from Provider B's domain and running 3rd party software components while provisioning the service to her customer.

The additional features of the proposed architecture cover the following areas: i) recursive orchestration API provided by NFVOs to other NFVOs; ii) customer contexts in the NFVO to expose different orchestration views to different customers based on policies; iii) recursive VNF template substitutions controlled by VNFMs that allow constructing VNFs based on other, potentially 3PP VNFs. The related workflows are the following.

The NFVO - NFVO API allows the subordinate NFVO to expose its abstract resource topology and capabilities towards customer NFVOs. The API may represent all or some of the domains and capabilities accessible via the particular provider. More exactly, based on the policies of the advertising provider, the exposed topology consists of one or more abstract nodes, virtual links and the list of VNF types available in the subordinate domains.

Using different orchestration views and contexts makes it possible to build a symmetric or peer-to-peer relationship between two providers with situational consumer/provider roles. The exposed resource views at the

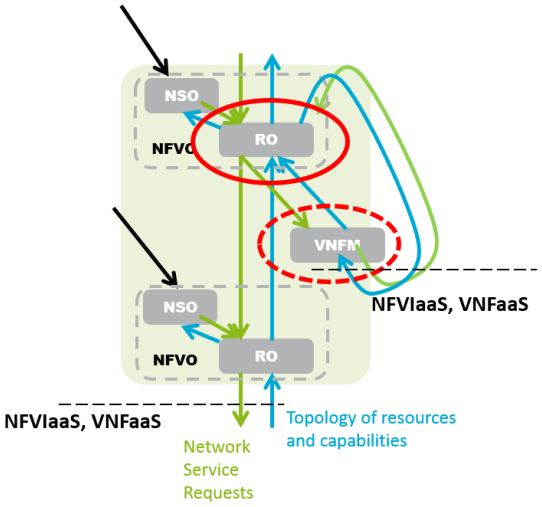


Fig. 3. NSO-VNFM-RO interfaces, workflows.

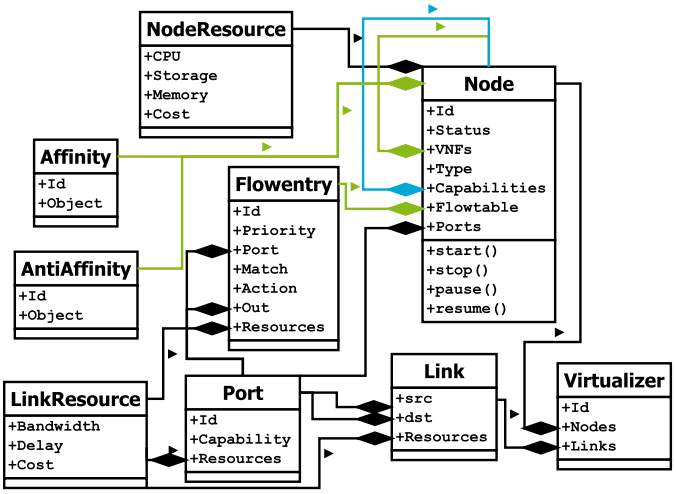


Fig. 4. Information model: A simplified view.

NFVOs should be configurable to reflect the desired aggregation level and information hiding policies of the operators.

Several orchestration hierarchies can be created on demand and exist in parallel.

When an orchestration request arrives, the NFVO checks local policies to identify the allowed resource domains for the request. This may be followed by VNF decomposition, managed by VNFMs, when compound VNFs are replaced by building block VNFs, which may refer to VNFs provided by underlying providers. The embedding algorithms are key components of the NFVO and enablers of several features expected from the federated orchestration system. Once the components are mapped to the abstract resource topology visible at the given layer, a second round of VNF subgraph substitutions is invoked for VNFs that require the mapping decisions as input.

Therefore, we distinguish two VNFM types corresponding to decomposition *before* and *after* the embedding. In the former case, the set of service components is embedding-agnostic, whereas in the latter case, the cardinality of functions depends on the topological embedding. For example, a distributed router/switch function needs to be deployed for each join and fork along the distribution network.

The dynamic behavior of the system is determined by the cooperation of components responsible for life-cycle management (LCM) and resource orchestration (RO). LCM functions shape the temporal aspects of a service while RO functions deal with their spatial aspects via resource assignment. On the one hand, Network Service Orchestration (NSO) is a typical end-to-end scoped LCM function encompassed by the NFVO. On the other hand, a VNFM is a component scoped LCM function which may decompose the component to a sub-service function chain. Fig. 3 shows the layering of the NSO, RO and VNFM. The bottom-up information flow corresponds to the topology of resources and capabilities, where capabilities are NFVIaaS and VNFaaS. Observe that the RO-RO and the RO-VNFM-RO interfaces are recurring, corresponding to incremental refinement of the embedding and to incremental decomposition of a service, respectively. In our design and implementation, all the interfaces crossing the red ovals in Fig. 3 (both solid and dashed) are the same. The NSO terminates this recursive pattern and provides an interface with the OSS/BSS systems.

C. Information Model

Central to the multi-provider hierarchical orchestration system is the information model that abstracts both i) the bottom-up network of compute resources and function capabilities, and ii) the top-down view of control over the virtualized infrastructure. This information model is used at the recurring resource control interfaces. The role one plays is situational, i.e., two peering providers will switch provider and consumer roles depending on who the tenant is that creates the service over the other's infrastructure. Since the ETSI NFV MANO architecture primarily focuses on single provider domain and virtualization control, they have not planned with the robustness necessary for inter-working among autonomous systems. Therefore, while retaining full compliance with ETSI Network Service offerings, we define our own information model, which supports an open ecosystem with multiple business actors.

In this heterogeneous setup, we strive to follow the success of IP's narrow waist concept, but this time for the management plane. Following basic economic trading notions, we designed an object-oriented information model (see Fig. 4) that makes the case of balanced assets and liabilities. The topology of resources and capabilities are the assets, while the topology of committed allocations are the liabilities: a *node* represents either an abstraction of node resources and capabilities or a VNF allocated on a node. Node objects have *ports* representing connection points; *links* connecting ports of different nodes define abstract 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 is 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, flowentries can be defined: we use *port - match - action* sets following the SDN design principle, but we support various technology specific mappings. Basic life-cycle management operations are contained in the *status* field of each node, such as create, start, stop, and pause.

The node object is named as Big Switch with Big Software (BiS-BiS). The task of service embedding is analogous to match the resource assignment from the northbound topology of allocations to the southbound topology of resources and capabilities. Multi-tenancy is supported by offering multiple northbound views over a shared southbound view: since the northbound view is not limited to a single BiS-BiS, an arbitrary topology of resources and capabilities can be defined according to standing business contracts, e.g., a control over East and West deployments, to be offered to the consumer. The flowrules may include $i)$ matching on input port, abstract tags or any other technology specific header fields, $ii)$ actions, such as output to port, push/pop abstract tags or any other technology specific packet manipulation. 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. Constraints associated with both the Node and Flowentry objects allow for pairwise (anti-)affinity "groups" or upper bounds on segment delay.

Comparing our information model with ETSI's, a service function chain mapped to a Single BiS-BiS view has its analogue with ETSI's VNF Forwarding Graph representation. Our model, however, $i)$ allows for the abstraction of an arbitrary topology of resources and capabilities, $ii)$ introduces typed VNFs, which correspond to VNFaaS, $iii)$ allows multiple capability instances of the typed VNF, hence connecting VNFaaS flavors to virtualization profiles, like latency of max rate through a VNF with given {cpu, mem, storage} profile; $iv)$ inherently supports basic LCM actions, so LCM functions can be transparently inserted in any orchestration "layer"; $v)$ allows full recursion, i.e., the northbound and southbound representations are the same for resource orchestrator and LCM components.

IV. Key Enabler: Our Embedding Algorithm

The most challenging part of the system is how to map the service components to resources. Our proposed online embedding algorithm operates in the NFVOs, which are connected in a multi-layer hierarchy, and supports several types of constraints, such as end-to-end QoS characteristics, cost limits and reliability requirements. The multi-layer distributed operation poses additional challenges to the algorithm and the framework. For example, certain constraints (e.g., anti-affinity) can be delegated via the orchestration hierarchy, but if the underlying NFVOs cannot fulfill those, a rollback mechanism should restore the previous consistent state of the overall system. In addition, the top level NFVO should be able to try other embedding options by the means of a multi-level backtracking mechanism. This section is devoted to this core component of our platform: we align the classical Virtual Network Embedding (VNE) problem to our environment, present the orchestration logic focusing on the advanced embedding features and the peculiarities of the multi-domain nature of the infrastructure, finally we analyze the efficiency of our algorithm.

TABLE I
Notations Used in Embedding Algorithm Description

Notation	Description
R= (V_R, E_R), S= (V_S, E_S)	Resource and service graph
$\mathcal{P}(S)$	Power set of set S
$e_j, j \in E_S \times V_S = \Psi_S$	Leg, the unit orchestration step
$\mu : \Psi_S \cup V_S \mapsto V_R$	Hosting node of service node
$\lambda : \Psi_S \cup E_S \mapsto \mathcal{P}(E_R)$	Hosting path of service link
$\mathcal{R}_{S,R}$, $\mathcal{F}_{S,R}$	Node resource and VNF types of R and S
$c_{V_R} : V_R \times \mathcal{R}_{S,R} \mapsto \mathbb{R}^+$	Resource capacity of a node in R
$c_{V_S} : V_S \times \mathcal{R}_{S,R} \mapsto \mathbb{R}^+$	Resource requirement of a VNF
$f_{V_R} : V_R \mapsto \mathcal{P}(\mathcal{F}_{S,R})$	Supported VNF types of a node
$f_{V_S} : V_S \mapsto \mathcal{F}_{S,R}$	Functional type of a VNF
$d_{E_R}, b_{E_R} : E_R \mapsto \mathbb{R}^+$	Delay and bandwidth characteristics of R links
$d_{E_S}, b_{E_S} : E_S \mapsto \mathbb{R}^+$	Delay and bandwidth requirement of S links
$\mathcal{A}_l^- : \Psi_S \mapsto \mathcal{P}(E_S)$	Link anti-affinity function
$\mathcal{A}_n^+, \mathcal{A}_n^- : V_S \mapsto \mathcal{P}(V_S)$	Node affinity & anti-affinity functions
$C^{E_S} \subset \mathcal{P}(E_S) \times \mathbb{R}^+$	Path delay requirements of S
$\mathcal{N} : \mathcal{P}(E) \mapsto \mathcal{P}(V)$	Nodes of a path (in resource or service graph)

A. Mathematical Problem Statement

The information model presented in Sec. III-C enables to describe the infrastructure in an abstract way spanning over the various technologically and/or administratively different domains. Each 5G operator represents their infrastructure with a set of BiS-BiS nodes. Both the inter- and intra-provider transport characteristics between the abstract BiS-BiS nodes are described by the link objects of the information model. This abstraction provides a graph-based representation of topology and capabilities (resource graph) $R = (V_R, E_R)$, and of service deployment requests (service graph) $S = (V_S, E_S)$ for our mathematical problem definition. Our problem formulation is a variant of VNE, which is a well studied research problem [24], [27].

We present an *online* version of the VNE problem, where only a single service graph is considered by (1)-(7).[1] The notations are summarized in Tab. I. The mapping structures μ, λ describe a mapping solution, where each service graph node V_S is mapped to a (possibly abstract) node in the resource graph V_R. The link mappings must be valid, i.e., their hosting paths must start and end at the hosts of their ends

[1]We omitted introducing node and link (anti-)affinity requirements and VNF hosting cost limits to our VNE problem definition to keep it simple.

as described by (1). The node mapping must respect the functional requirements of the VNFs $i \in V_S$ as required by (2). Constraint (3) ensures that node capacity requirements mapped to a resource node $u \in V_R$ do not exceed the total capacity of the node for each resource type $\mathcal{R}_{S,R}$. Link-wise delay requirements must be respected by the link mapping function λ for each VNF connection $(i,j) \in E_S$ as stated by (4). Constraint (5) defines the set of all VNF connections using a substrate network connection as $M(u,v)$. This set is used to summarize all the bandwidth capacity requirements $b_{E_S}(i,j)$, which must be upper bounded by the resource link's bandwidth capacity $b_{E_R}(u,v)$.

$$\forall (i,j) \in E_S : \mu(i) = \lambda(i,j).first \quad \text{and} \quad \mu(j) = \lambda(i,j).last \tag{1}$$

$$\forall i \in V_S : f_{V_S}(i) \in f_{V_R}\big(\mu(i)\big) \tag{2}$$

$$\forall u \in V_R, \forall r \in \mathcal{R}_{S,R} : \sum_{\{i|\mu(i)=u,\ \ i \in V_S\}} c_{V_S}(i,r) \leq c_{V_R}(u,r) \tag{3}$$

$$\forall (i,j) \in E_S : \sum_{(u,v)\in\lambda(i,j)} d_{E_R}(u,v) \leq d_{E_S}(i,j) \tag{4}$$

$$\forall (u,v) \in E_R, \quad M(u,v) := \{(i,j) | (u,v) \in \lambda(i,j) \text{ and } (i,j) \in V_S\} : \sum_{(i,j)\in M(u,v)} b_{E_S}(i,j) \leq b_{E_R}(u,v) \tag{5}$$

$$\forall (p, D_p) \in \mathcal{C}^{E_S} : \sum_{(i,j)\in p} \sum_{(u,v)\in\lambda(i,j)} d_{E_R}(u,v) \leq D_p \tag{6}$$

$$min_{\mu,\lambda} \sum_{e_j, j \in \Psi_S} \textsc{CalcObjectiveValue}\Big(\lambda(e_j,j), \mu(e_j,j), e_j, j\Big) \tag{7}$$

In addition to link-wise delays, we extend the VNE problem with path delay requirements $\mathcal{C}^{E_S}$, which contain maximal allowed latency on multiple consecutive VNF connections of E_S. These type of path delay constraints are highly motivated by real world applications, as presented earlier in Sec. II-B. The optimization goal of our *online* variation of VNE will be detailed during the embedding algorithm description, for now we just give a general objective function which minimizes the sum of the objective value of each VNF $j \in V_S$ and the adjacent service graph connection $e_j \in E_S$.

Most of the practically interesting variants of the VNE problem are known to be NP-hard and strongly inapproximable [31]. Our formulation introduces more constraints, such as the path delay requirement, so the same observations about complexity apply to our VNE formulation.

B. Basic Operation of the Embedding Algorithm

The embedding algorithm computes the mapping to the underlying abstract resources exposed by partner providers. The service and resource graph models enable efficient and versatile embedding algorithms to be implemented. Our orchestration engine runs a heuristic-guided greedy backtracking search on the resource graph structure. An overview on the formal description of this approach is shown in Alg. 1. Refer to Tab. I for a summary of the notations.

Algorithm 1 Overview of the Embedding Algorithm Returns a Set of Complete Mapping Structures of Service Graph S to Resource Graph R

```
procedure MAP(S, R) → μ, λ
    Ψ_S ← ORDERLEGSFORMAPPING(V_S, E_S)
    while ∃e_j, j ∈ Ψ_S where ∄μ(e_j, j) or ∄λ(e_j, j) do
        while MAPONENF(e_j, j) not successful do
            e_j', j' ← GETBACKTRACKOPTION(e_j, j)
            UNDOGREEDYMAPPING(e_j', j')
            e_j, j ← e_j', j'
        end while
    end while
    return μ, λ
end procedure
```

Algorithm 2 Details of the Objective Value Calculation for Greedily Choosing the Locally Most Preferred Resource Node and Hosting Path for a Leg. Returns a Real Value, Which Is Used to Sort the Hosting Options of a Leg

- Ω_x, ρ_x are the value and weights of the bandwidth, resource, latency, cost components, denoted by $x \in \{bw, res, lat, cost\}$ respectively.
- ω_r are the weights of node resource component r.
- ξ_1, ξ_2 are the weights of the latency components.

```
procedure CALCOBJECTIVEVALUE(e_j, j, p_{u⇝v}, v)
    Ω_bw ← GETAVERAGEPATHBWUTIL(p_{u⇝v})
    Ω_res ← Σ_{r∈R_{S,R}} ω_r GETNODERESUTIL(v, r)
    Ω_lat ← ξ_1 DISTANCEFROMLASTHOST(p_{u⇝v}) + ξ_2 DIRECTTOWARDSENDOFPATHLATENCY(e_j, j, v, μ)
    Ω_cost ← GETCOSTOFLEGHOST(p_{u⇝v}, v, e_j, j)
    return Σ_{x∈{bw,res,lat,cost}} ρ_x Ω_x
end procedure
```

An elementary mapping step is the greedy allocation of a VNF and an adjacent service graph link, called leg (e_j, j), onto a hosting (virtual) substrate node and path. An embedding order among these elementary steps is calculated by the function ORDERLEGSFORMAPPING(V_S, E_S). In case a greedy step is not able to find a suitable host, while respecting all service graph requirements, the most recent greedy step is undone by freeing the temporarily reserved resources. The orchestration engine yields an embedding solution, when all elements of the service graph have been successfully mapped respecting each aspect of the requirements or refuses the request altogether. Alg. 1 returns the embedded solution μ and λ, which give the hosting node of a VNF and the hosting path of a service graph link respectively. More details on the ORDERLEGSFORMAPPING(V_S, E_S) and the greedy backtracking, without considering advanced features and multi-domain support, is presented in our earlier publication, which also compares the algorithm's performance to that of Integer Programming methods [30].

In each greedy step, the hosting substrate path and node pair with the lowest objective function value is chosen for mapping, and the next couple of best ones (controlled by the branching factor) are stored for possible later

exploration. The details of how the embedding possibilities are sorted is shown in Alg. 2. Besides the resource availability and greedy search directing objective function components, GETCOSTOFLEGHOST$(p_{u\rightsquigarrow v}, v, e_j, j)$ calculates the cost of using a hosting resource graph path $p_{u\rightsquigarrow v} \in \mathcal{P}(E_R)$ and host $v \in V_R$ for (e_j, j).

The search space size of the greedy backtracking 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). The weighted sum of the objective's four components enables us to tune the algorithm to multiple application scenarios. A given setting for the weights of latency subcomponents ξ_i, resource type subcomponents ω_r and objective components ρ_x provide a fully specified optimization goal for the mathematical problem statement in (7). These parameters enable our multi-domain orchestrator to be versatile: (i) optimizing for bandwidth utilization on infrastructure connections, (ii) distributing node resource utilization among operators, (iii) providing high service acceptance for delay critical applications, (iv) minimizing administrative costs of routing and VNF hosting, or (v) arbitrary superposition of multiple operating policies from the various involved entities.

Our proposed orchestration engine can be applied on each abstraction layer of the recursive infrastructure. Due to the hidden information of an underlying operator resources (e.g. if a single aggregated BiS-BiS view is shown from a domain), it is possible that a selected embedding solution on the abstract view of an upper layer proves to be infeasible on a lower abstraction layer. In this case, the orchestrators of different layers communicate this failure to the appropriate domains, which undo the failed (partial) service instantiation. Our algorithm supports this scenario efficiently: if a lower abstraction layer failure notification arrives, the greedy backtracking search continues from the latest solution, eliminating the need to calculate the whole orchestration process from scratch.

C. Advanced Embedding Features: End-to-End Delays, Node and Link (Anti-)Affinities

We consider path delay support and affinity type requirements as advanced orchestration features, because they are generally not included in VNE problem definitions. Moreover, to the best of our knowledge no other VNE algorithm considers anti-affinity requirements on service graph paths. We include these features in our proposed multi-domain orchestration engine.

Apart from basic service graph requirement of node and link capacities, VNF type constraints and link-wise delay requirements, our orchestration system supports maximum end-to-end allowed delay requirements on service graph paths, as stated by Constraint (6). The greedy orchestration steps can be controlled by the value of the latency preference Ω_{lat} component and its weight ρ_{lat}, as shown in Alg. 2. If such end-to-end requirement is given for orchestration, a delay budget is allocated for all affected BiS-BiS nodes, whose orchestrators over lower level resource abstractions receive this delay budget as input end-to-end requirements. This approach provides the multi-domain, multi-operator support of guaranteed end-to-end path delays over a unified abstract interface. An example for the application of this feature is from Fig. 1, where Op. 1 needs to allocate a delay budget for the Factory Provider's orchestrator to ensure she can meet the strict delay requirement needed between the robot at the factory site and the control logic at Op. 1's edge data center.

Node affinity requirement, denoted by $\mathcal{A}_n^+$, is a binary relation between VNFs of the service graph, which requests the affected VNFs to be placed on the same hosting nodes. This requirement always needs to be forwarded to the lower orchestration layers until the physical layer is reached. The capability of an infrastructure node for receiving VNF affinity requirements is described by the *Capabilities* field of an abstract node (see Fig. 4). In contrary, the binary relation of node anti-affinity requirement, denoted by $\mathcal{A}_n^-$, specifies the affected NFs to be placed on different abstract and physical nodes. The first procedure of Alg. 3 shows the details of the node affinity and anti-affinity criteria compliance at each greedy orchestration step. In our orchestration system, the node anti-affinity criteria are preferred to be resolved in the orchestration hierarchy as high as possible. In case it is not possible due to other requirements ruling out the desired mapping possibilities, the resolution of node anti-affinity criteria can be delegated to abstract BiS-BiS nodes.

Algorithm 3 Details of the MAPONENF(e_j, j) Function's Node Affinity and Anti-Affinity and Link Anti-Affinity Criteria Compilation. Returns Whether a Resource Path and Node of a Leg Complies to All Affinity-Like Criteria

```
1: procedure COMPLYNODECRITERIA(j, v, A_n^+, A_n^-)
2:     for all (A, ⊛) ∈ {(A_n^+, ≠), (A_n^-, =)} do
3:         for all k ∈ A(j) do
4:             if ∃μ(k) and μ(k) ⊛ v then
5:                 return False
6:             end if
7:         end for
8:     end for
9:     return True
10: end procedure
11:
12: procedure COMPLYLINKCRITERIA(e_j, j, p_{u⇝v}, v, A_l^-)
13:     for all e ∈ A_l^-(e_j, j) do
14:         if ∃λ(e) then
15:             if λ(e) ∩ p_{u⇝v} ≠ ∅ then
16:                 return False
17:             end if
18:
19:             if N(p_{u⇝v}) ∩ N(λ(e)) ≠ ∅ then
20:                 return False
21:             end if
22:         end if
23:     end for
24:     return True
25: end procedure
```

The lower abstraction level orchestrators get the delegated anti-affinity criteria as input, and follow the same approach. If any node anti-affinity criteria cannot be resolved on the selected search space throughout the whole orchestration hierarchy, a failure is propagated to upper orchestrators.

We have extended the anti-affinity requirement for service graph paths as well: logical links of the resource graph can be grouped as link anti-affinity groups (LAAFFG). Binary relations among LAAFFGs define that all included VNFs and links of a LAAFFG must be mapped to resources independent of the hosting elements of the opposing LAAFFG (see PID_A, PID_B and their connections in Fig. 1). The link anti-affinity structure is denoted by $\mathcal{A}_l^-$, and it identifies a set of service graph edges, which need to be in anti-affinity relation with a given leg $e_j, j \in \Psi_S$. The second procedure of Alg. 3 shows the details of how the link anti-affinity, i.e., the hosting resource graph node and link independence requirement, is enforced for one greedy orchestration step. In line 18, $\mathcal{N}(\lambda(e))$ stores the *momentarily* used infrastructure nodes by a service graph link's $e \in E_S$ path. An empty intersection of $\mathcal{N}(\lambda(e))$ and the nodes of the current path ensures that the same nodes from R are not used for two links in anti-affinity relation. The anti-affinity relations must always be forwarded to the lower orchestrators, because the resource independence must be respected even through the hierarchical layers, as demonstrated by Enterprise Customer 2's request in Fig. 1.

D. Complexity Analysis

In our previous work [33], we analyzed the complexity of our basic algrithm, now we focus on the extended version, supporting affinity and anti-affinity constraints as well.

Theorem 1: The computational complexity of our basic embedding algorithm is

$$\mathcal{O}\left(\max\left\{|V_R|^2|E_R|, b\log b\right\}\frac{b^{k+1}}{b-1}\left\lceil\frac{|V_S|}{k}\right\rceil\right).$$

Theorem 1 shows the polinomial runtime in the input sizes $R = (V_R, E_R), S = (V_S, E_S)$ of the basic version of the orchestration logic. The b fix parameter is the branching factor of the greedy backtrack search, which restricts how many possible hosts for a leg (e_j, j) can be stored, while k fix parameter is the backtracking limit. This makes our embedding solution fixed parameter tractable for backtrack parameters b and k. Since the algorithm has been extended by the support for node and link (anti-)affinity requirements, we revise this analysis accordingly.

Theorem 2: The computational complexity of complying the node and link (anti-)affinity criteria is

$$\mathcal{O}\Big(|E_R|^2 + |E_R||V_R| + |V_S|\Big).$$

Proof: The innermost *for* cycle of the first procedure of Alg. 3 runs at most $|\mathcal{A}|$ iterations, which needs to be executed for both the affinity and anti-affinity node criteria. Therefore its runtime can be upper bounded by a constant factor of the two structures' summed size, which can be upper bounded by the VNF count: $\mathcal{O}(|\mathcal{A}_n^+| + |\mathcal{A}_n^-|) = \mathcal{O}(|V_S| + |V_S|) = \mathcal{O}(|V_S|)$. The second procedure runs at most $|\mathcal{A}_l^-|$ iterations (upper bounded by the link count of R), which means always evaluating the existential quantifier ($\exists$) as true, assuming the worst case runtime. Evaluating the intersections of the anti-affinity constraints for both the current hosting node and path structures can be upper bounded by the node and link counts of the infrastructure respectively: $\mathcal{O}(|\mathcal{A}_l^-|(|V_R| + |E_R|)) = \mathcal{O}(|E_R|(|V_R| + |E_R|)) = \mathcal{O}(|E_R|^2 + |E_R||V_R| + |V_S|)$. This yields the overall runtime complexity of Alg. 3. □

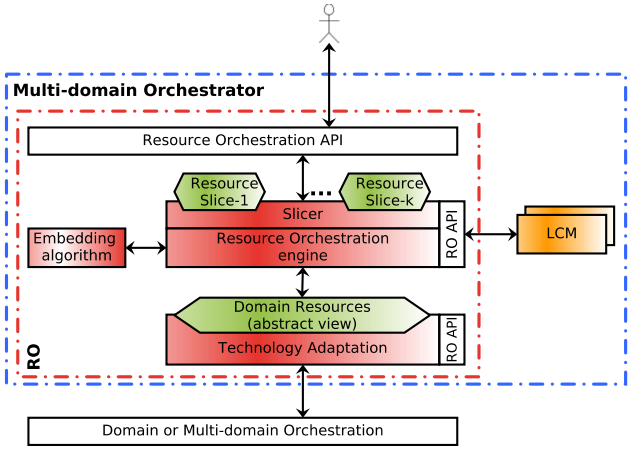


Fig. 5. Software architecture of our orchestrator.

Theorem 3: The overall complexity of our embedding algorithm respecting node and link (anti-)affinity requirements is

$$\mathcal{O}\left(\max\left\{|V_R|^2|E_R| + |E_R|^2 + |V_S|, b\log b\right\}\frac{b^{k+1}}{b-1}\left\lceil\frac{|V_S|}{k}\right\rceil\right).$$

Proof: As argued in our paper [33], due to the backtracking search approach MAPONENF(e_j, j) is used at most $\frac{b^{k+1}}{b-1}\left\lceil\frac{|V_S|}{k}\right\rceil$ times. The two procedures of Alg. 3 are both evaluated for each MAPONENF(e_j, j) execution, so its complexity is added to the first argument of the $max\{\}$ function in Theorem 1. Adding the statement of Theorem 2 to our basic algorithm's complexity, having $|E_R||V_R|$ dominated by $|V_R|^2|E_R|$, leads to the overall complexity of our feature-rich multi-domain embedding algorithm. □

This statement shows that the runtime is polinomial in the input size with low exponents. Supporting the node and link (anti-)affinity requirements introduces acceptable overhead.

V. IMPLEMENTATION

In this section we present our proof-of-concept prototype implementing the relevant parts of the proposed orchestration system, which is released as open-source. The main functional blocks are shown in Fig. 5.

A. Resource Orchestrator

Our Resource Orchestrator (RO) encompasses and coordinates multiple components, as it is shown in Fig. 5. In general, it is responsible for exposing different virtual resource views upwards and for satisfying service deployment requests. The requests are expressed on the high-level virtual views (Resource Slices) and mapped onto the full domain view,

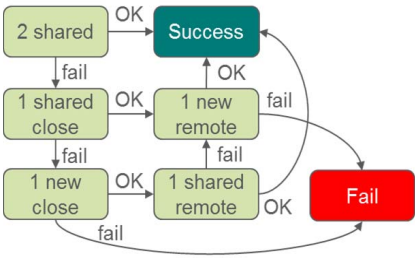


Fig. 6. Finite state machine in the robot control VNFM.

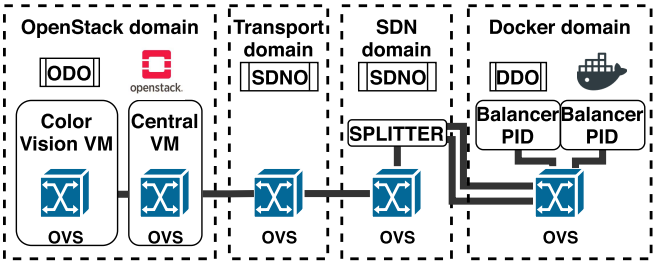


Fig. 7. Realization of the robot control service.

which encompasses the underlying resources and topologies. In Fig. 5 hexagon (green) boxes correspond to resource views while filled rectangles (red and also orange ones) indicate orchestration or control related elements. During the orchestration workflow, RO engine invokes the Embedding algorithm module, which performs the mapping of the service requests to the available resources, according to the configured algorithm and policies. The result describing the full deployment is then sent to the Technology Adaptation component. RO also provides a domain-agnostic resource abstraction and virtualization for different resources, technologies or administrative domains. By these means, RO acts as a multi-domain, multi-vendor and technology-independent controller entity. Before and after the embedding step, the interaction with LCM modules is also coordinated by RO.

Slicer is an integrated part of the RO and its role is fourfold: i) it introduces multi-tenancy by configurable northbound views, programmable resource aggregation and information hiding; ii) it enforces operational 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; iv) it loops in LCM functions (specific VNFMs): forwards requests and combines responses into the end-to-end service function chain before passing it to the embedding logic. The LCM support in the slicer is recursive: a service component decomposed/substituted by the LCM logic may contain service components provided by further LCM logic registered with the slicer.

B. Lifecycle Manager

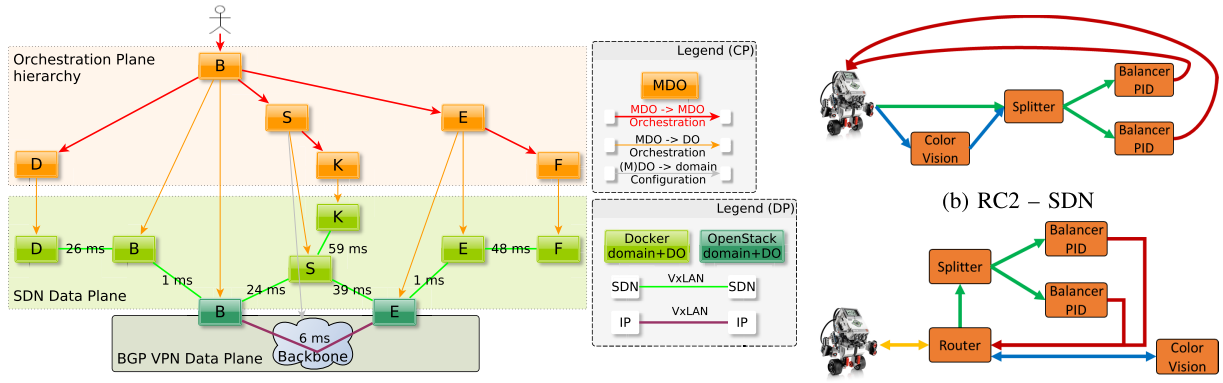
We implemented two LCM/VNFM modules showcasing the two different types of VNFMs (activated before/after the embedding). The first one handles the reliable robot control (RC2) service presented in Sec. II-B. If the Slicer detects RC2 compound VNF in the request, it forwards that to the registered LCM module. In our current implementation, VNF sharing is supported and preferred. More specifically, the VNFM tries to reuse already deployed components (PID controllers in the use-case) and starts a *trial and error* operation phase following the finite state machine shown in Fig. 6. In this phase, VNFM and RO are cooperating in order to find a feasible resource assignment. VNFM is iterating among the possible embedding states and initiates RO for trying to realize the resource orchestration for the current state. After each trial and error cycle, RO gives back a success/fail report to the VNFM regarding the current state. As a first option, VNFM tries to reuse two available PID VNFs, more exactly, a closer one meeting the strict latency requirement and a backup/remote one. If it fails, VNFM tries to share only one of the VNFs and requests a new instance from the other one. When sharing is not feasible, new instances are deployed.

Our second VNFM controls the deployment of a distributed router (dR) VNF (also shown in the example of Sec. II-B). It is activated after the embedding process and based on the mapping decisions, the result is updated accordingly. Specifically, multiple dR elements are added if more underlying domains are involved, or dR is replaced to technology specific configurations depending on the capabilities of the given domains. For example, in an SDN capable domain abstract flowrules are added, whereas for a legacy IP/MPLS domain BGP VPN configuration is provided. The stitching points between different technology domains are also considered by the implemented VNFM.

C. Domain Orchestrators: OpenStack, Docker, SDN

In order to evaluate our concept in realistic scenarios, the adaption to today's VIMs is crucial. On the one hand, we have developed a dedicated library (Virtualizer) fostering the adaptation of different technologies. The library makes it possible to add a northbound interface to available VIMs, which is compliant with our resource control interface. On the other hand, we have implemented domain orchestrators to widely used VIMs.

As OpenStack is the most widely used open-source cloud "operating system", its integration in our framework is inevitable. Our domain orchestrator provided for OpenStack is referred to as OpenStack Domain Orchestrator (ODO). ODO exposes a northbound interface, which is used to interact with upper layer NFVOs following the information model presented in Sec. III-C. A dedicated resource orchestrator module is the core component, which parses the given configuration files, maintains the current topology and the database of supported VNFs and manages OpenStack through its REST APIs. ODO supports two operation modes. The first one is an SDN compatible mode, which can handle requests containing VNFs connected by SDN flowrules. Here, all VNFs (VMs) contain a wrapper function with an included Open vSwitch (OVS) which allows to handle control plane massages coming from the orchestrator. Furthermore, the wrapper is responsible for



(a) Orchestration hierarchy in our sandbox

(b) RC2 – SDN

(c) RC2 with router

Fig. 8. Our sandbox environment and the evaluated robot control services (RC2, see Sec.II-B).

creating VXLAN tunnel endpoints and virtual interfaces in the deployed instances. The other operation type of ODO supports only "legacy IP" network connections. Using this configuration option, the VNFs are deployed as simple VMs with no extra wrapper functions and distinct neutron networks are created for VNF interfaces. Each neutron network is connected to at least one router to provide the connection between the deployed VNFs. We also support the interconnection between SDN and legacy IP domains. Our solution assumes BGP-based IP VPN networks between domains realized by the BagPipe driver. BGP VPNs are deployed and managed by domain operators, in particular to manage Route Target identifiers that control the traffic isolation between different VPNs. Additionally, in the SDN domain, we need a special proxy VM, responsible for traffic encapsulation and decapsulation between the two technological domains.

An example realization is presented on Fig. 7, where the solid lines show the flows of data. In an SDN compatible OpenStack domain a special VM appears, called "central VM", whose main goal is to provide data plane connection between the components of each domain. Similarly to ODO, Docker Domain Orchestrator (DDO) was implemented to support Docker-based domains managing light-weight containers instead of VMs. A DDO provides an upper layer above the Docker and handles VNF deployment through the Docker API. The data plane connections are realized by VXLAN tunnels,for which endpoints are configured in the Docker's OVS.

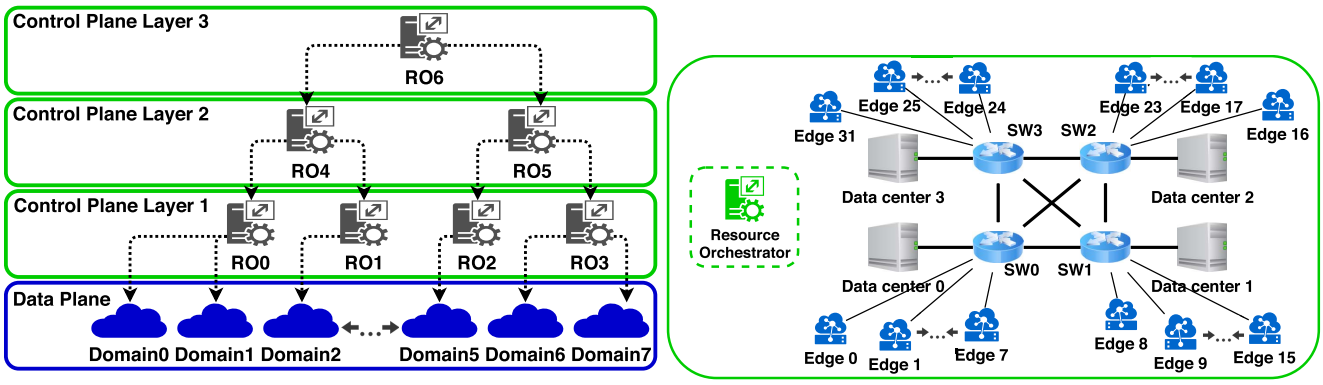
Finally, we implemented an SDN Orchestrator (SDNO), which is capable of managing traffic forwarding rules in OpenFlow networks. This solution is used in two different ways: i) managing transport domains; ii) deploying VNFs as OpenFlow flowrules. In the former case, the domain only transmits the data plane traffic through its network, and SDNO manages the necessary flowrules in an OVS. In the latter case, SDNO deploys those VNFs, which can be realized with OpenFlow flowrules, e.g., traffic splitter/duplicator or a data plane IP router. Although, our SDNO is not a fully-fledged OpenFlow controller, it supports both software-, and hardware-based OpenFlow devices. We use technology specific adaptation to implement end-to-end connectivity over domains with different network service capabilities. Such technology 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 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.

VI. Evaluation

In this section we demonstrate that the orchestration system, with all its elements presented in the previous section, scales well and does not raise significant performance issues. We underpin this statement with experiment results that we obtained on a large distributed testbed with a high number of deployed services. The results show that the time the control plane actions take is orders of magnitude lower than what is inevitably spent with firing up data plane components.

In order to validate the automated end-to-end orchestration of envisioned multi-provider 5G services in real-life situations, we built a sandbox that includes several administrative domains, called sandbox islands, across Europe. These islands are hosted by our collaborating partners including academia, operators and vendors. Each island runs an instance of the proposed prototype with all the components (LCM, RO, Slicer) locally, as well as infrastructure domains with domain orchestrators, e.g., Network (SDN), Docker or OpenStack domains, with their corresponding domain orchestrator.

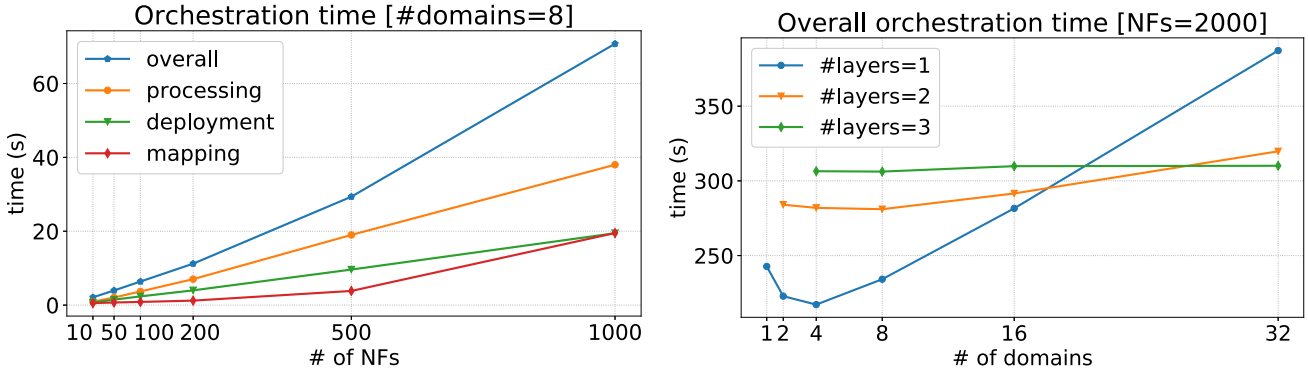
We organize the islands in a situational hierarchy as it is shown in Fig. 8a in an anonymized way. A set of Tier 1 providers (B, E and S) are connected to a backbone network with BGP VPN support which is under the control of provider S. Other providers, i.e., D, F and K, are connected to the federation via Tier 1 operators in data plane, while the control plane hierarchy is created independently. We assume that operator B is the customer-facing provider sitting at the top of the hierarchy. Resource control interfaces between multi-domain orchestrators (MDOs) are indicated by red arrows, while the control relation between MDO and DOs are shown by orange ones. In the data plane, green lines correspond to SDN overlay, whereas purple lines identify legacy IP connections. In the



(a) Test infrastructure

(b) Evaluated Edge/Cloud topology

Fig. 9. Test scenarios and topologies used in control plane experiments.



(a) Scalability with service requests

(b) Scalability in multi-layer hierarchy

Fig. 10. Control plane experiments on scalability with emulated data plane and orchestration hierarchies.

TABLE II
SANDBOX EXPERIMENTS

	RC2 – SDN	RC2 – router
mapping	0.25 sec	0.32 sec
deployment (DO+VIM)	110.23 sec	217.96 sec
overall time	112.67 sec	219.2 sec

figure, we show only data plane latency characteristics among participating resource domains.

A. Sandbox Experiments

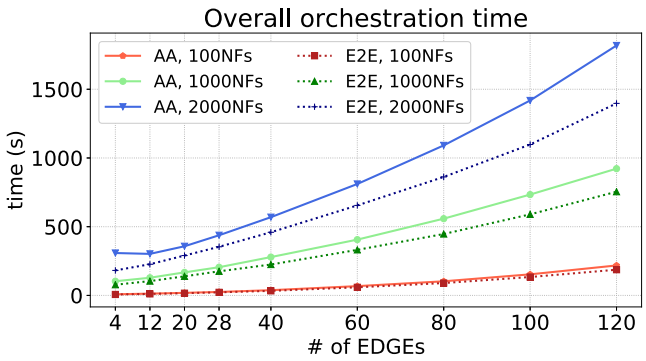
In our sandbox environment, we conducted several experiments with different versions of the Robot Control service envisioned in Sec. II-B (RC2). Two examples are shown in Fig. 8: Fig. 8b corresponds to a service with SDN overlay between the components, while the service shown in Fig. 8c requests a distributed router to establish the connectivity among the constituent VNFs (managed and unmanaged ones).

The runtime of different operation phases for two selected illustrative experiments are presented in Tab. II. In these experiments, we deployed services shown in Fig. 8b and Fig. 8c, respectively. The latter one requires all relevant features and involves all prototype elements, thus show poorer performance. The results confirmed that the overall deployment time is mainly affected by the performance characteristics of the VIMs (starting VMs and containers).

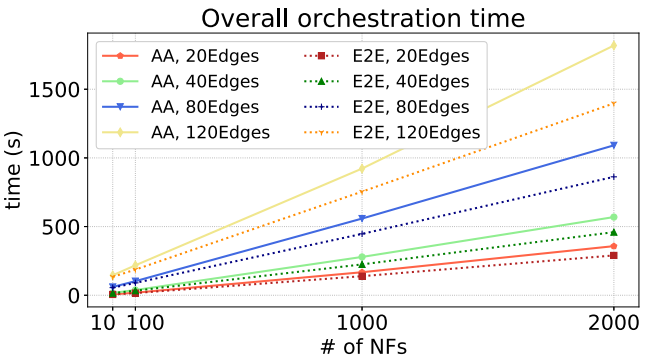
B. Control Plane Experiments

For large scale test scenarios we have implemented random service graph generators which create arbitrarily sized inputs based on the given parameters. The number of constituent NFs of generated requests is at most 10 and they are divided approximately equally among the underlying domains.

Fig. 9a shows the general structure of our multi-layered control plane test scenarios. Data plane resources are managed by a given number of Resource Orchestrator instances (8 in the figure) organized into an arbitrary control plane hierarchy (3-layered binary tree in the figure). Fig. 10a presents the scalability characteristics of the control plane operations in terms of the size of the service requests. In this experiment, we constructed a flat orchestration hierarchy consisting of a multi-domain orchestrator and 8 different domain orchestrators under that. Service requests including increasing number of VNFs from 10 to 1000 were sent in respective batches. The plots indicate the runtime of different phases of the control plane process, such as mapping (embedding algorithm), deployment related functions, processing tasks (manipulating



(a) Scalability with resource domains



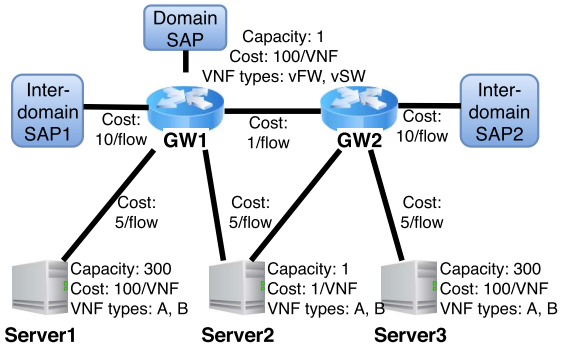
(b) Scalability with service requests

Fig. 11. Control plane experiments on emulated Edge/Cloud infrastructure.

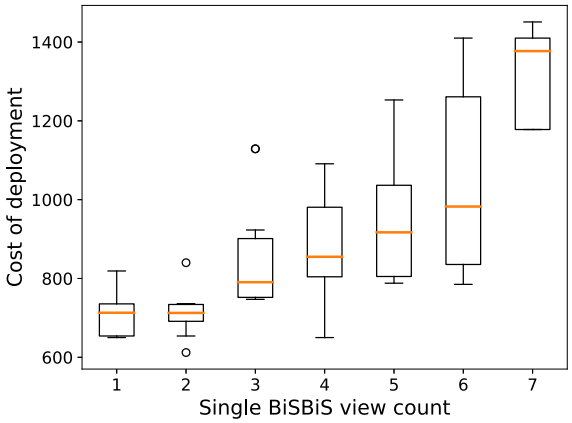
the data models) and the overall time. Even in case of the largest request, the overall operation time was below 80 sec and the results show promising scalability properties.

Fig. 10b depicts overall runtimes for a control plane implementing a multi-layer hierarchy of multi-domain orchestrators on top of increasing number of domain orchestrators. The three curves correspond to a hierarchy of 1, 2 and 3 distinct layers, respectively. Increasing the number of orchestration layers introduces a bigger overhead of overall orchestration runtime, but after a certain number of domains, improving performance can be observed as the complex request is decomposed and processed in parallel by the involved orchestrators.

We also evaluated Edge/Cloud scenarios and the results of a selected topology shown by Fig. 9b are presented. Here, the core full-mesh network connects 4 cloud servers which provides the majority of the available computational resources and the distant edge nodes that are capable of running a limited number of NFs. All the participating components are managed by one orchestrator instance to form a single flat CP hierarchy. Fig. 11a illustrates the impact of the number of simultaneously controlled domains tested by different size of 5G-aware services. Here, two types of service requests were evaluated: one with end-to-end delay requirements comparable to the diameter of the topology and another one containing anti-affinity constraints between dedicated NF pairs. The overall runtimes of these services are comparable to each other and show that services with anti-affinity relations require slightly more time for the orchestrator to deploy. Finally, Fig. 11b shows the performance in terms of the complexity of service requests. The presented results confirm the polynomial scaling properties of the orchestration system.



(a) The topology of a single domain



(b) Total cost of deployment by increasing the number of hidden topology domains

Fig. 12. Experiment on the evaluation of resource abstraction: infrastructure of an operator on the left, total deployment cost results on the right.

C. Experiments on the Effects of Infrastructure Abstraction

We performed experiments to evaluate the effect of network abstraction on the cost of the service placement. In total 8 domains are simulated with the topology shown in Fig. 12a, which are connected by data plane links in a circular fashion, i.e., each domain's *Inter-domain SAP1* is connected to the neighbor's *Inter-domain SAP2*. Each domain is managed by an RO, and multi-domain orchestration is performed by a top level RO, managing all 8 domains. The embedding algorithm of each RO is configured to minimize cost, i.e., for the objective component $\Omega_{cost} = 1$, while other components have 0 weights in Alg. 2.

The ROs orchestrating the individual infrastructure domains represent infrastructure providers, who can decide the level of infrastructure abstraction to be shown to the multi-domain RO, i.e., the customer-facing provider in this case. In our experiments, each domain RO can show their network topology with all details (see Fig. 12a), or they can aggregate everything into a single BiS-BiS node. No partial information sharing, e.g.,

disclosing arbitrary fraction of the infrastructure information, is allowed. The multi-domain RO collects the information shared by the domain ROs, and makes placement decisions, which need to be further orchestrated by ROs that share only the aggregated single BiS-BiS information, to map the received service components to the hidden resource elements. These ROs have to set their prices for the BiS-BiS nodes they present, which tells the multi-domain RO how much it needs to pay for hosting a single VNF. A domain owner might set the price based on many different factors, such as business strategy, techno-economic analysis, game theoretical strategy, etc. For the sake of simplicity, in our experiments each domain uses the average cost of their node resources as the BiS-BiS node's price. Capacities and VNF-type capabilities are simply aggregated by summation and union, respectively.

A service request is received by the top level RO, containing 6 VNFs (2x type A, 2x type B, 1x vFW and 1x vSW) with unit capacity requirement, and connecting the *domain SAP*s of two distinct domains by 8 service graph edges (flows). The experiments examine a corner case: with cost and capacity values shown in Fig. 12a, the BiS-BiS view of any domain hides the capacity bottleneck on its *Server2* of capacity 1. Fig 12b shows the total cost of service deployment as more and more ROs decide to show only the aggregated BiS-BiS information instead of all details of their infrastructure. The experiment for each case is repeated 10 times, while the single BiS-BiS domains are randomly selected. The median of the total deployment cost monotonously increases, resulting in a 40% difference between from having a sole single BiS-BiS to reaching 6 among the 8 underlying domains. In the extreme case of having overwhelming majority (7 out of 8) of the operators disclosing aggregate views, the deployment cost doubles. Based on our simulations, we conclude that bottlenecks are worth advertising to higher level orchestrators in order to avoid costly deployments.

VII. Conclusion

This paper summarizes our steps that we made in the direction of realizing the long-awaited 5G vision. We used a top-down approach: started from the requirements of future online applications, we derived the most important principles, which then guided our careful design of a multi-provider cross-domain service creation platform. We produced a novel resource model yielding many benefits, the most important one being the recursiveness that we can exploit in the provider-customer hierarchy. We planned the architecture of the system with a revisited separation of functionalities that are responsible for static resource allocations and the dynamic management of services' life-cycles. The proposed notion of resource slicing complements state-of-the-art network slicing designs by creating wholesale offerings at the providers. In order to prove that our system design is valid, we implemented all the main components of the platform, conducted experiments and showed the benefits of our approach. Specifically, the embedding algorithm that is the core intelligence of our proposed framework, proved to be scalable, hence ready for the orchestration of large-scale future services over 5G.

References

[1] *Apache CloudStack*. Accessed: Jan. 31, 2018. [Online]. Available: https://cloudstack.apache.org

[2] *Ciena Blue Planet MDSO*. Accessed: Jan. 31, 2018. [Online]. Available: http://www.blueplanet.com/products/multi-domain-service-orchestration.html

[3] *Cloudify: Cutting Edge Orchestration*. Accessed: Mar. 14, 2020. [Online]. Available: https://cloudify.co

[4] *Docker: A Better Way to Build Apps*. Accessed: Jan. 31, 2018. [Online]. Available: https://www.docker.com

[5] *Industry 4.0: The Fourth Industrial Revolution—Guide to Industrie 4.0*. Accessed: Jun. 18, 2018. [Online]. Available: https://www.i-scoop.eu/industry-4-0/

[6] *Kubernetes: Open-Source Production-Grade Container Orchestration*. Accessed: Jan. 31, 2018. [Online]. Available: https://kubernetes.io

[7] *Linux Containers*. Accessed: Jan. 31, 2018. [Online]. Available: https://linuxcontainers.org

[8] *Metro Ethernet Forum: Third Network*. Accessed: Jan. 31, 2018. [Online]. Available: https://www.mef.net/third-network-vision-work-of-the-mef

[9] *NGENA: The Next Generation Enterprise Network Alliance*. Accessed: Jan. 31, 2018. [Online]. Available: http://www.ngena.net/

[10] *ONAP: Open Network Automation Platform*. Accessed: Feb. 2, 2020. [Online]. Available: https://www.onap.org

[11] *ONOS: Open Network Operating System*. Accessed: Jan. 31, 2018. [Online]. Available: http://onosproject.org

[12] *OPEN-O: Open Orchestrator Project*. Accessed: Jan. 31, 2018. [Online]. Available: http://www.openhub.net/p/open-o

[13] *OpenBaton: An NFV-MANO Framework*. Accessed: Jan. 31, 2018. [Online]. Available: http://openbaton.github.io/

[14] *OpenStack: Open Source Cloud Computing Software for Private and Public Clouds*. Accessed: Jan. 31, 2018. [Online]. Available: https://www.openstack.org

[15] *OpenStack Tacker*. Accessed: Jan. 31, 2018. [Online]. Available: https://wiki.openstack.org/wiki/Tacker

[16] *OPNFV: Open Platform for NFV*. Accessed: Jan. 31, 2018. [Online]. Available: https://www.opnfv.org

[17] *OSM: Open Source NFV Management and Orchestration (MANO)*. Accessed: Jan. 31, 2018. [Online]. Available: https://osm.etsi.org

[18] *The CORD Platform: Central Office Re-Architected as a Datacenter*. Accessed: Jan. 31, 2018. [Online]. Available: https://www.opennetworking.org/cord/

[19] *The OpenDaylight Platform (ODL): Open Source SDN Platform*. Accessed: Jan. 31, 2018. [Online]. Available: http://www.opendaylight.org

[20] *World's First 5G Federated Network Slicing Grants Global Reach*. Accessed: Jan. 31, 2018. [Online]. Available: https://www.ericsson.com/en/press-releases/2017/2/worlds-first-5g-federated-network-slicing-grants-global-reach

[21] J. Quittek, P. Bauskar, T. BenMeriem, A. Bennett, and M. Besson, "Network functions virtualisation (NFV)–management and orchestration," ETSI, Sophia Antipolis, France, Tech. Rep. GS NFV-MAN 001, Dec. 2014.

[22] J. Xia *et al.*, "Network functions virtualisation (NFV)–acceleration technologies-report on acceleration technologies & use cases," ETSI, Sophia Antipolis, France, Tech. Rep. GS NFV-IFA 001, Dec. 2015.

[23] X. Haitao *et al.*, "Network functions virtualisation (NFV) release 3; management and orchestration; report on architecture options to support multiple administrative domains," ETSI, Sophia Antipolis, France, Tech. Rep. GS NFV-IFA 028, Dec. 2017.

[24] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves, "On the computational complexity of the virtual network embedding problem," in *Proc. 7th Int. Netw. Optim. Conf. Electron. Notes Discrete Math. (INOC)*, vol. 52, Jun. 2016, pp. 213–220.

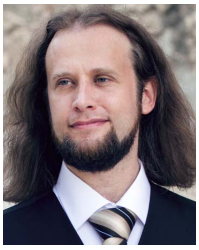
[25] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *Proc. 11th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2015, pp. 50–56.

[26] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *Proc. IEEE 28th Conf. Comput. Commun. (INFOCOM)*, Apr. 2009, pp. 783–791.

[27] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, 4th Quart., 2013.

[28] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 94–100, May 2017.

[29] C. Fuerst, S. Schmid, and A. Feldmann, "Virtual network embedding with collocation: Benefits and limitations of pre-clustering," in *Proc. IEEE 2nd Int. Conf. Cloud Netw. (CloudNet)*, Nov. 2013, pp. 91–98.
[30] B. Nemeth, B. Sonkoly, M. Rost, and S. Schmid, "Efficient service graph embedding: A practical approach," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2016, pp. 19–25.
[31] M. Rost and S. Schmid, "Charting the complexity landscape of virtual network embeddings," in *Proc. IFIP Netw. Conf. (IFIP Netw.) Workshops*, May 2018, pp. 1–9.
[32] P. Rost *et al.*, "Network slicing to enable scalability and flexibility in 5G mobile networks," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 72–79, May 2017.
[33] B. Sonkoly, M. Szabo, B. Nemeth, A. Majdan, G. Pongracz, and L. Toka, "FERO: Fast and efficient resource orchestrator for a data plane built on Docker and DPDK," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 243–251.
[34] H. Zhang, N. Liu, X. Chu, K. Long, A.-H. Aghvami, and V. C. M. Leung, "Network slicing based 5G and future mobile networks: Mobility, resource management, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 138–145, Aug. 2017.



Balázs Sonkoly (Associate Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science from the Budapest University of Technology and Economics (BME) in 2002 and 2010, respectively. He is an Associate Professor with BME. He is also the Head of MTA-BME Network Softwarization Research Group. His current research interests include cloud/edge/fog computing, NFV, SDN, and 5G. He has participated in several EU projects (FP7 OpenLab, FP7 UNIFY, and H2020 5G Exchange) and national projects. He was the Demo Co-Chair of ACM SIGCOMM 2018, EWSDN'15,'14, and IEEE HPSR'15.

Róbert Szabó received the MBA, M.Sc., and Ph.D. degrees in E.E. from the Budapest University of Technology and Economics (BME). Since joining Ericsson in 2013, he was the Technical Coordinator of the EU-FP7 Unifying Cloud and Carrier Networks (UNIFY) Integrated Project from 2013 to 2016. He was the Project Coordinator of the H2020 5G-PPP 5G Exchange (5GEx) Innovation Action from 2015 to 2018. He is currently working with distributed/edge cloud, zero touch automation, and network function virtualization. He worked as an Associate Professor with BME, where he was the Deputy Head of the Department of Telecommunications and Media Informatics from 2008 to 2010. He is a Principal Researcher with Research Area Cloud Systems and Platform, Ericsson Research.

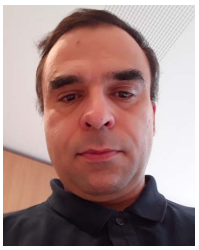
Balázs Németh (Graduate Student Member, IEEE) received the M.Sc. degree from the Budapest University of Technology and Economics (BME) in 2016. He is currently pursuing the Ph.D. degree in network softwarization with special focus on orchestration algorithms and next generation network models. He is an Industrial Ph.D. student with BME in cooperation with Ericsson Research. He was a Computer Science Engineer in info-communication specialization with BME. He has been working on orchestration algorithms for the H2020 5G-PPP 5G Exchange (5GEx) Project.

János Czentye (Graduate Student Member, IEEE) received the M.Sc. degree (Hons.) in networks and services in 2014. He is currently pursuing Ph.D. degree with the Budapest University of Technology and Economics. Since 2014, he has been with the Department of the Telecommunications and Media Informatics contributing in several research projects and gained wide knowledge about SDN, NFV, and microservice technologies. His current Ph.D. research focuses on cloud-native service modeling and provisioning.

Dávid Haja (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the Budapest University of Technology and Economics. He is a member of High Speed Networks Laboratory, Department of Telecommunications and Media Informatics. His main research interests include edge computing, software-defined networking (SDN), network function virtualization (NFV), and resource orchestration.

Márk Szalay (Student Member, IEEE) is currently pursuing the Ph.D. degree with High Speed Networks Laboratory, Budapest University of Technology and Economics. His main research interests include hardware (router/switch/NIC) design, network programming, software-defined networking, and network function virtualization.

János Dóka (Graduate Student Member, IEEE) received the M.Sc. degree from the Budapest University of Technology in 2020. He is a member of High Speed Networks Laboratory, Department of Telecommunications and Media Informatics and the MTA-BME Network Sofwarization Research Group. His current research interests include network function virtualization and serverless computing.



Balázs P. Gerő (Associate Member, IEEE) received the M.Sc. degree in computer science from the Budapest University of Technology and Economics (BME) in 1998. He joined Ericsson Research in 2000. His research interests include a variety of fields, such as network management, transport network protocols, software-defined networks, and data centers. He is currently active in the orchestration system design of large scale distributed/edge cloud systems.

Dávid Jocha (Member, IEEE) received the M.Sc. degree from the Budapest University of Technology in 2002. He joined Ericsson Research. He has been active in Ethernet performance measurement, radio network optimization, and SDN related topics. His current interests include orchestration and cloud technologies.

László Toka (Member, IEEE) received the Ph.D. degree from Telecom ParisTech in 2011. He worked at Ericsson Research from 2011 to 2014, then, he joined the academia with research focus on software-defined networking, cloud computing, and artificial intelligence. He is an Assistant Professor with the Budapest University of Technology and Economics, the Vice-Head of High Speed Networks Laboratory, and a member of both MTA-BME Network Softwarization and MTA-BME Information Systems Research Groups.