Time-Efficient Distributed Virtual Network Embedding For Round-Trip Delay Minimization

Ioannis Dimolitsas[a,*], Dimitrios Dechouniotis[a], Symeon Papavassiliou[a]

[a]*School of Electrical and Computer Engineering, National Technical University of Athens, 9, Iroon Polytechniou Street, Athnes, GR 15780, Greece*

Abstract

Virtual Network Embedding (VNE) aims at determining the placement of virtual services both internally in the edge/cloud infrastructure (EC) and distributed, among distinct ECs, under several resource and network constraints. With regard to Network Function Virtualization technology, a virtual service is referred to as Virtual Network Function (VNF). Modern 5G applications related to the Internet of Things (IoT) are deployed as interconnected VNFs with specific execution sequence in the form of a Service Function Chain (SFC), composing a Virtual Network (VN), and the execution result has to be returned back to the original end-device. Thus, the minimization of the round-trip delay is of major importance to guarantee the performance requirements of IoT-based applications. Additionally, in such a dynamic environment, the re-optimization of VNE is often required to meet the network delay requirements due to end-device mobility, and other orchestration constraints related to the limited edge resources. In this paper, we propose a distributed VNE (DVNE) approach, which focuses on minimizing the round-trip delay. Initially, we introduce an algorithm, namely *sV-VNM*, that associates every VNF of the VN with a set of candidate ECs that can be hosted onto. Given this initial mapping, an efficient path-based algorithm, namely *kSP-DVNE*, is proposed to provide the DVNE solution, in polynomial time. Evaluation results show that the sV-VNM algorithm maps the VN onto the ECs, with decreased resource utilization within the infrastructure, while the *k*SP-DVNE achieves optimal or near optimal solutions regarding the round-trip delay minimization, with a significant reduction in the execution time, compared to existing relevant approaches in the literature.

Keywords: Virtual Network Embedding, Network Function Virtualization, Edge Computing, Service Function Chain, Delay Minimization

1. Introduction

Modern 5G applications rely on various Internet of Things (IoT) devices, often generating vast amount of data. Usually, these IoT devices have limited computational resources and energy capacity, thus, they are not able to locally perform complex processing. On the other hand, Edge Computing promises to mitigate the above restriction by providing computing infrastructure in the proximity of the end-users. Furthermore, processing at the network edge results in reduced network latency comparing with processing on traditional cloud resources. However, contrary to the coarse cloud case, the edge resource management must be fine-grained due to the limited edge resources and the strict time constraints of IoT applications [1]. In this context, many resource orchestration platforms have been developed that rely on either virtual machines (VM) or containers, such as OpenStack [2] and Kubernetes [3], and they provide several functionalities about service embedding, resource allocation and networking.

Network Function Virtualization (NFV) architecture enabled the running of network services as VMs on common hardware and therefore replaced any proprietary middle-box. An application consists of individual services, namely Virtual Network Functions (VNFs). A Service Chain Function (SFC) includes several VNFs with predefined execution order and specific computing and network requirements. SFCs can either be unidirectional or bidirectional. Unidirectional SFCs require traffic to go through the ordered VNFs in one direction, while bidirectional SFCs need a symmetric path, where the VNF instances are the same in both directions. [4] The typical Virtual Network Embedding (VNE) problem focuses on embedding the SFC on the substrate network. Under this setting, the network traffic is directed from an ingress VNF to the egress one, while the response traverses the same path in the opposite direction. ETSI NFV [5] and ETSI Multi-Access Edge Computing (MEC) [6] are the two dominant reference architectures that are used for the realization of the VNE in cloud and edge infrastructure.

However, with the advent of 5G and IoT technologies, the formulation of the VNE requires two key modifications. At first, the early definition of VNF strictly referred to network services (routing, firewall, and load balancing, etc.). Nowadays, the VNF definition should be more application-oriented and include any type of service. Secondly, for IoT applications, which usually include online machine learning processing [7], the forward traffic consists of raw data, while the backward traffic includes results and models that must be sent back to the end devices that generate the network traffic. Additional time- or

*Corresponding author

Email addresses: jdimol@netmode.ntua.gr (Ioannis Dimolitsas), ddechou@netmode.ntua.gr (Dimitrios Dechouniotis), papavass@mail.ntua.gr (Symeon Papavassiliou)

mission-critical applications, in which IoT devices send and receive data to/from the edge/cloud, are related to Healthcare (Remote Healthcare Monitoring, Remote Surgery, Connected Ambulances) [8, 9, 10], vehicular communication and autonomous driving [9], and online gaming [8, 11]. This possibly requires different forward and backward paths for the SFC. This type of SFC, where the result is required to be transmitted back to the end-devices, is defined as a hybrid-SFC [4, 12] (h-SFC).

Edge Computing provides geographically spread small-scale resources that create a distributed cloud environment with multiple stakeholders [13]. This enables the creation of Network Service Marketplaces, which allows subscribers to act both as service providers and consumers [14]. In this complex scenario, an already deployed VNF can be shared among different tenants in order to reduce the allocated resources in an Edge Cloud (EC) and optimize the exchanging traffic between them. Under this setting, a SFC includes pre-installed VNFs that must be taking into consideration in the VNE solution. Furthermore, the minimization of network delay is the dominant architectural goal of Edge Computing to cope with the IoT proliferation [11]. The limited computing capabilities of IoT devices require a lot of tasks to be performed fully on edge computing infrastructures or partially via computational offloading. In parallel, the reduction of round-trip delay is crucial for ensuring of the strict quality of service (QoS) requirements of such applications [8]. Furthermore, to deal with the dynamic conditions of IoT and Edge Computing, the multi-objective approaches are usually computationally expensive [15] and may not always result in near-optimal solutions with respect to all objectives. On the contrary, computational-light and effective distributed embedding mechanisms, which prioritize the reduction of round-trip delay, can compute near-optimal solutions in a real-time fashion. Focusing on IoT applications deployed on distributed edge infrastructure, this work proposes a distributed virtual network embedding mechanism that takes into account the above characteristics of the SFC and the distributed multi-stakeholder edge environment. More specifically the scientific key contributions of this article are the following,

- A heuristic that undertakes the optimal initial mapping of parts (VNFs) of a Virtual Network (VN) within an EC infrastructure considering the pre-deployed shared-VNFs, focusing on minimizing the infrastructure's resource utilization.

- Assuming that the forward and the backward paths of a VN are different, the proposed shortest-path-based algorithm finds the Distributed Virtual Network Embedding (DVNE) solution that minimizes the round-trip delay.

- The numerical results illustrate that the proposed distributed virtual network embedding algorithm computes the optimal or near optimal solution, especially combined with the initial VN mapping heuristic, in limited time compared to relevant existing state of the art studies.

The rest of the article is structured as follows. Section 2 presents the related work, while Section 2 provides the fundamental definitions of the DVNE problem. Section 4 formulates the Intra-EC VN mapping problem and the delay-aware DVNE problem and the next section presents the proposed algorithms for solving both problems. Section 6 presents the performance evaluation of the proposed DVNE solution and compares it against other related studies demonstrating its benefits and tradeoffs. Finally, Section 7 draws the conclusions and highlights possible future work directions.

2. Related Work

This section presents a comprehensive overview of the most relative studies in literature. For presentation purposes these studies are classified in three main categories, (i) shared VNF-based VNE approaches, (ii) distributed VNE approaches, and (iii) shortest path-based VNE approaches.

Shared-VNFs host common services, that can be consumed simultaneously by different tenants. A primary requirement of shared-VNF is the isolation between different SFCs that share such VNF. Aligned with ETSI NFV and ETSI MEC reference architectures, the MESON platform provides a secure mechanism for communication between slices of different tenants in order to save computing resources within an EC and reduce outgoing network traffic [16]. Shared-VNFs could be consumed by multiple slices by leveraging the Open Source MANO (OSM) [17] network slice orchestration capabilities [18]. Papadakis et al. [19] proposed a blockchain-based mechanism for cross-service communication that provides registration, searching, leasing, and billing functionalities over multiple ECs in the context of Network Service Marketplaces. Similarly, based on distributed ledger technology, the FENDE marketplace relies on shared-VNFs and facilitates the subscribers to deploy tailor-made SFCs over multi-cloud infrastructure [20]. Edge Computing enables the deployment of network services in small-scale infrastructure close to the end-users. Contrary to the cloud case, this infrastructure is geographically spread, and therefore, the VNE problem has been transformed to the DVNE one. For MEC/Cloud environment, Zheng et al. [12] proposed a hybrid form of SFC with dissimilar forward and backward paths. This study focused on the latency minimization and it proposed a backtracking-like method on an augmented graph to select the shortest path. Considering a distributed cloud environment with three types of stakeholders (i.e., subscribers, service provides, and infrastructure providers), the authors in [21] formulated the DVNE problem as an Integer Linear Programming to both maximize the number of accepted SFC requests and satisfy the subscribers' requirements. Towards this direction, a pre-processing stage aims at rejecting those requests that are infeasible to be accomplished by the system, defining incompatibilities between VNFs and data centers and defining subscriber's preferences that are used in the objective function of the optimization model. Pei et al. [22] formulated the DVNE problem as a Binary Integer Programming model. Subsequently, they proposed two algorithms for minimizing the embedding costs. Initially, the VNFs instances are placed by a shortest-path algorithm in a multi-layer graph. Then, the placed VNFs are released using their utilization rate and a threshold according

Table 1: Comparison of related studies.

Categorization	Related Studies										Proposed Study
	[16]	[19]	[20]	[12]	[21]	[22]	[23]	[24]	[25]	[26]	
Shared VNF-based VNE	✓	✓	✓	x	x	x	x	✓	x	x	✓
Distributed VNE	x	x	x	✓	✓	✓	✓	x	✓	✓	✓
Intra-EC VNE	✓	✓	✓	x	✓	x	x	x	x	x	✓
Path-based VNE	x	x	x	✓	✓	✓	✓	✓	✓	x	✓
Round-Trip Delay	x	x	x	✓	x	x	x	x	x	x	✓
Compute Resources	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Network Bandwidth	x	x	✓	✓	✓	✓	✓	✓	✓	✓	✓
Order Constraint	x	x	x	✓	✓	✓	✓	✓	✓	x	✓

to workload variation. Constraints that refer to delay, reliability, mobility and battery requirements, of mobile end-devices which act as an extension of the cloud and edge computing infrastructure are taken under consideration for the VNF placement by the authors in [23]. The problem is formulated as a cost-minimizing VNF placement optimization. A heuristic based on the fractional optimal solution of a bin packing variant is proposed to obtain the near optimal VNF placement solutions, while ensuring scalability in terms of reducing the convergence time, as occurred from simulations under a mobile robots scenario.

For either VNE or DVNE problem, many studies proposed shortest path-based models that allow computing a near optimal solution with low complexity. The authors in [24] focused on the trust-aware VNE problem. For each VNF and network edge, they assumed trustworthiness requirements and formulated a path-based model to integrate network policies. In order to enhance the scalability of the model, they included the k-shortest paths of an augmented graph to find the optimal solution. The authors in [25], proposed a constrained shortest path algorithm to deal with the exponential nature of the VNE problem. They introduced the *Neighborhoods Method*, which utilizes dynamic programming and branch-and-bound exhaustive search, while search space reduction techniques are, also, considered. This method achieves a theoretical reduction of computational complexity compared to alternative exhaustive search solutions. Evaluation results certify the scalability and flexibility of the proposed algorithm when compared to other path finding-based algorithms. The VNE problem in SDN is discussed in [26]. The authors provided a multi-objective optimization formulation of the VNE problem, towards, firstly, the minimization of the network load and secondly, the maximization of the embedding reliability, under the constraints of virtual network requirements and the resource characteristics of the substrate network. The problem is separated into two corresponding sub-problems. For solving these sub-problems, a two-stage VNE algorithm is proposed using slight modifications for each case. The authors assigned performance metrics to both virtual nodes and substrate nodes to perform the mapping greedily, while the virtual links are embedded by applying the Dijkstra's algorithm in the weighted graph representation of the substrate network, to determine the shortest path between the source and the destination substrate node of one virtual link. Then, examining the distance between the feasible solutions and the locally optimal solutions, they formulated a single-objective optimization problem and solved it to obtain the global VNE strategy.

Table 1 summarizes the achieved goals of the aforementioned studies and highlights the differences with the proposed solution. Contrary to our work, the shared VNF-based studies [16, 19] focused mainly on the service discovery and the establishment of cross-slice communication in a single EC, while the Intra-EC VNE is performed by default component of the VM orchestrator (i.e., OpenStack). On the other hand, FENDE marketplace [20] focused on the synthesis of the customized SFC, and the VNE solution is handled by a local Virtualized Infrastructure Manager. Regarding the distributed VNE approaches, Cappanera et al. [21] proposed an optimization model with high execution time that increases rapidly as the number of EC increases. Furthermore, the proposed Intra-EC VNE solution is performed in an aggregated fashion and does not consider any co-location criterion. In addition, the intra-EC part of the VNE problem is only considered as a constrained variable in the respective formulations in [22, 23], while the models contemplate the end-to-end SFC path embedding, which is not directly apply for hybrid-SFCs towards round-trip delay minimization. The same stands for the path-based solutions in [24, 25], where search-space reduction techniques are applied, but mainly links capacity constraints are taken into account, while in [26] the VNE is performed based on link and network switch modeling, with abstractly described computing constraints, in the absence of the execution order constraints, in accordance with the SFC specifications [4]. Zheng et al. [12] deals with the hybrid-SFC nature of VNE, with round-trip delay minimization. However, the described solution focuses on constructing the DVNE path, regardless of how the service functions are placed within the ECs. Aspiring to overcome all aforementioned limitations, our article focuses on providing distributed VNE solutions with the objective of minimizing the round-trip delay, in a time efficient manner that will be able to run online, and deal with the dynamic nature of modern 5G virtualized applications. The latter is motivated by the fact that reconfiguration of the embedding

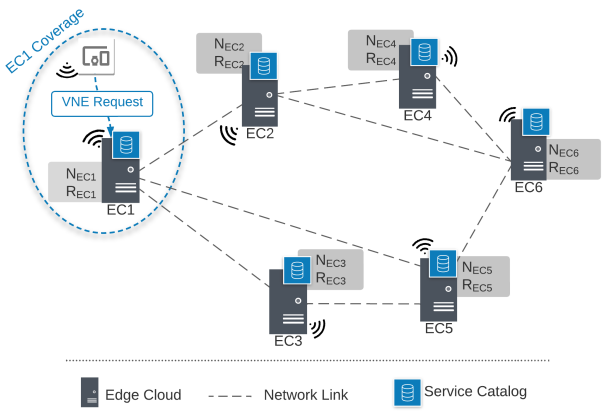


Figure 1: Edge Network Topology.

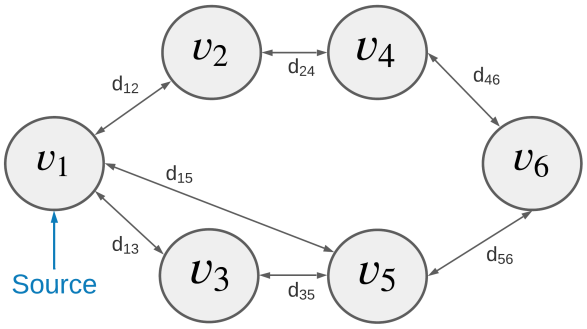


Figure 2: Edge Network's Graph Representation.

is frequently required due to end-device mobility or workload management and resource allocation constraints from the edge infrastructure perspective.

3. System Modelling

As described earlier, most of the works in the literature focus on the standard unidirectional SFC, where the traffic follows a path from source to destination, through the specified VNFs in accordance with their execution order [4]. However, in particular IoT scenarios, the application execution outcome is required to be returned to the end-device. Following the definition in [4, 12] for a h-SFC, a forward path to a specific destination point, which could be on the Cloud, for processing the forward traffic, and a backward path, through which the processing result is returned back to the end-device, are defined. These paths are not necessarily identical [12]. Also, the destination VNF of an h-SFC could not be specific or fixed, and so the forward and backward path compose a continuous data stream among the defined sequence of the VNFs [4]. Due to our focus on the deployment of virtualized applications at the edge, we extend the h-SFC concept to a more holistically defined schema and we refer to that as Virtual Network (VN).

Definition 1. *A Virtual Network (VN) is a set of VNFs and virtual links, which is determined by a specific execution sequence, and the corresponding computing and network resource demands. The execution outcome is required to be returned to the source of the network traffic (end-device).*

The rest of this section describes the system modeling and the parameters of the DVNE problem in detail. Figure 1 illustrates the Edge Network topology, which consists of geographically distributed EC infrastructures. End-users can submit their VNE requests in any approximate EC. As mentioned above, in the context of Edge Computing, we assume that the services of a Virtual Network are deployed as VNFs in the form of SFC. A VNE request dictates the placement of the individual VNFs and includes both *(i)* new VNFs for deployment and *(ii)* shared VNFs that are already deployed in specific ECs.

3.1. Substrate Network Model

We illustrate the Edge Network as an undirected Graph $G = (V, E)$. The graph representation of the Edge Network of Fig. 1 is illustrated in Fig. 2, while the set of network model parameters is shown in Table 2. In order to avoid confusion, we state that with the word *edge*, we refer to Edge Computing artifacts, and with the word *link* to the network connection between the EC nodes or VN nodes, as well as the connections between the vertices of the respective graph representation of them. The set $V = \{v_1, \ldots, v_n\}$ corresponds to the nodes of G, where $v_i \in V$ represents the EC_i of the Edge Physical Network. The network links between the ECs v_i, v_j of the Edge Network constitute the set E, where $e_{ij} = (v_i, v_j) \in E$. The available computing resources, R_v, and the provided shared VNFs, $\mathcal{N}_v$, are determined as the main attributes of an EC $v \in V$. Besides that, d_{ij} denotes the delay of the network link e_{ij}. As it is shown in Fig. 1, the EC1 (v_1), in which the VNE request is initially submitted, is considered as the source node, denoted by v_0 and it is where the end-device is connected.

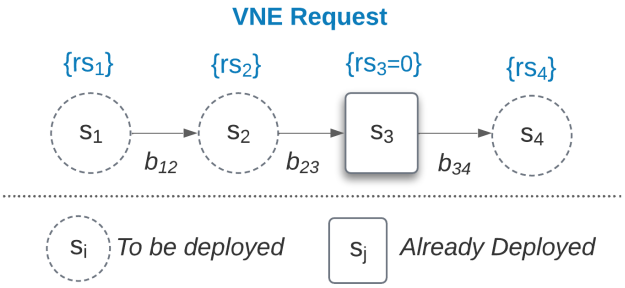


Figure 3: Virtual Network Embedding Request Model.

3.2. VNE Request Model

The end-user submits a VNE request in any EC $v \in V$. The end-device which is the source of the network traffic is denoted as s_0. Let $G_s = (V_s, E_s)$ be the graph representation of the VNE request. The set V_s denotes the VNFs that constitute the VNE

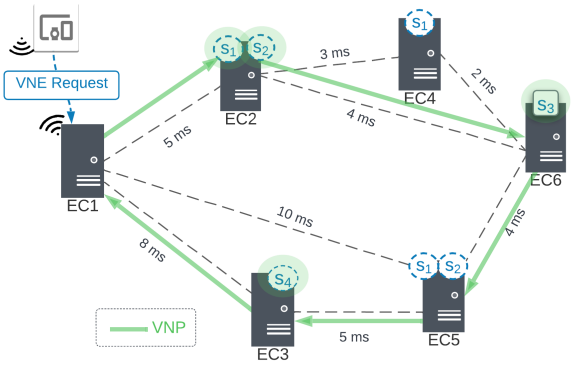


Figure 4: An example of Distributed VNE.

request. Every element of V_s is a tuple $\langle s_i, rs_i \rangle$, where s_i represents the i^{th} VNF of the VN, and rs_i its resource requirements; as the shared VNFs within the VN are already deployed in an EC, it stands that if s_i is a shared-VNF then $rs_i = 0$. Furthermore, the E_s is the set that contains the virtual links between the VNFs of the request. Each virtual link $\left(s_i, s_j\right) \in E_s$ is associated with its residual bandwidth demand, denoted as b_{ij}. Figure 3 illustrates a VNE request, which consists of four VNFs. The s_3 is a shared-VNF and as we mentioned above, its resource requirements are considered equal to zero regarding the VNE. An example of an embedding solution for this VNE request on the aforementioned edge network is illustrated in Fig.4. As shown, the VNE request is submitted in the EC1, as the end-device operates in its coverage. For this VNE request, the VNFs s_1, s_2 are embedded in the EC2, the s_3 in the EC6, and the s_4 in the EC3. In this case, the EC1 does not host a VNF of the request; it is just considered as the ingress node of the solution, which includes the necessary paths for the VNE.

4. DVNE Problem Formulation

The discussed DVNE problem is broken down into two subproblems. The first one concerns an initial mapping of the VN, or parts of it, in all ECs of the Edge Network. Then, the construction of the VN is performed in a distributed manner, where each VNF $s \in V_s$ has to be deployed in one of the candidate ECs selected on the mapping of the first phase.

4.1. Initial Intra-EC VN mapping

Aiming at providing a DVNE solution, an initial mapping of VN parts in the respective EC infrastructures of the Edge Network is performed. With this capacity, each VNF will have a set of candidate ECs that could be embedded onto. Towards the initial mapping of a VN within the ECs, several parameters must be taken under consideration. In particular, shared-VNFs are pre-deployed in specific ECs, and the residual computing and network capacity of the ECs is constrained in terms of the available resources. Taking that into account, a primary fact is that

Table 2: System's key notations.

Parameter	Definition
Edge Network Parameters	
G	Edge Physical Network graph
V	The set of EC nodes $v_1, v_2, \ldots$ of G
E	The set of links (v_i, v_j) of G
$\mathcal{N}_v$	Available shared VNFs in node $v \in V$
R_v	Available Computing Resources of node $v \in V$
d_{ij}	The network delay of the link $e(v_i, v_j) \in E$
VNE Request Parameters	
G_s	NSE request service graph
V_s	The set of VNFs in G_s
E_s	The set of virtual links (s_i, s_j)
s_i	The i^{th} VNF in set V_s
rs_i	The demanded resources of VNF s_i
b_{ij}	The bandwidth demand of link $(s_i, s_j) \in E_s$
Problem Formulation Parameters	
v_0	The EC that the end-device is connected
u_i	A host EC v for the VNF s_i, $i = 1, \ldots, m$
$\mathcal{P}$	A DVNE solution
$w(v_i, v_j)$	The shortest path's cost between v_i, v_j
D_{rt}	The round-trip delay of a DVNE solution $\mathcal{P}$

a VNE request will have to be partitioned, initially, into more than one sub-VNs. In essence, this first phase is actually formulated as a typical SFC embedding problem in a cloud infrastructure [16]. In this paper, we refer to this problem as Intra-EC VN mapping. The goal of this initial phase is the minimization of computing and network resource utilization through optimized mapping of sub-VNs within an EC infrastructure. Moreover, several works in the literature highlight the importance of co-location between pairs of service nodes (VNFs) of a network slice embedding, which is another version of the DVNE problem, in a cloud data center [27], especially regarding the network latency and bandwidth, as well as other resource allocation parameters (CPU utilization, Inter-Rack network traffic within the EC infrastructure). Therefore, the main objective regarding the initial Intra-EC VN mapping is to achieve a high co-location ratio between adjacent VNFs of a VNE request. Then, the initial mapping is used on the solution of the DVNE subproblem.

4.2. Delay-Aware Distributed VNE

The initial Intra-EC VN mapping actually computes multiple embedding alternatives of entire (or part of) VNE requests in various ECs. The goal of the second phase is to provide a DVNE solution that minimizes the round-trip delay. Assuming that an end-device generates the incoming traffic of the VN and the response of the corresponding services (VNFs) is returned to this device, we determine the *VN path (VNP)* as a circuit with source (and sink) node the EC that the end-device is connected. Given a VNE request with $V_s = \{s_1, s_2, \ldots, s_m\}$ and the source

of the network traffic s_0 (end-device), we define VNP as:

$$VNP = \{s_0, s_1, s_2, \ldots, s_m, s_0\}. \quad (1)$$

Towards a path-based DVNE formulation, the following definitions are required; *(i)* A DVNE solution is denoted by $\mathcal{P}$ and determines the sequence of the host ECs for every VNF of a VNE request, starting and ending in the EC v_0 that the end-device s_0 is connected,

$$\mathcal{P} = \{u_0, u_1, \ldots, u_m, u_{m+1}\}, \quad (2)$$

where $u_0 = u_{m+1} = v_0$, and u_i corresponds to the EC v that is the host of a VNF s_i in the solution $\mathcal{P}$, for $i = 1, \ldots, m$. So, $u_i = v \in V$. It is worth mentioning that the node v_0, which is mapped onto the variables u_0 and u_{m+1} of the solution $\mathcal{P}$, is the EC to which the end-device is connected and does not necessarily have to host a VNF of the request. Thus, it could be considered only as the first hop of the VNP, without the need of allocating resources. It stands that $\mathcal{P} \subset V$, while its cardinality is: $|\mathcal{P}| = m + 2$. Let the binary variable $\rho_i(v)$ indicates whether an EC v is a host for a VNF s_i in a DVNE solution $\mathcal{P}$,

$$\rho_i(v) = \begin{cases} 1, & \text{if } u_i = v \\ 0, & \text{otherwise} \end{cases}, \; i = 1, \ldots, m. \quad (3)$$

(ii) The cost of the shortest path between two ECs v_a, v_b in the graph representation G of the Edge Network, corresponds to the network delay between the respective ECs and is denoted by $w(v_a, v_b)$. The round-trip delay of a solution $\mathcal{P}$ is denoted by $D_{rt} = \sum_{i=0}^{m} w(u_i, u_{i+1})$. Taking into account the aforementioned system modeling, our objective is to compute a DVNE solution that minimizes the round-trip delay D_{rt}:

$$\min_{P^*} \sum_{i=0}^{m} w(u_i, u_{i+1}) \quad (4a)$$

s.t.:

$$\sum_{v \in V} \rho_i(v) = 1, \; \forall i \in \{1, \ldots, m\} \quad (4b)$$

$$\sum_{i=1}^{m} \rho_i(v) rs_i \leq R_v, \; \forall v \in V \quad (4c)$$

$$w(u_i, u_{i+1}) < \infty, \; \forall i \in \{0, 1, \ldots, m\} \quad (4d)$$

The constraint (4b) indicates that for every solution $\mathcal{P}$, each service s_i has a unique hosting EC v, for a specific VNE request. The constraint (4c) guarantees that the requested resources by VNFs do not exceed the resource availability of every EC, while (4d) ensures the existence of a path between the ECs that compose the DVNE solution. In essence, (4d) focuses on the connectivity of the graph that corresponds to the solution P; the paths that are going to be used in the solution must exist, also, in the Edge Network Topology graph representation. In the following, the proposed solution is thoroughly described and consists of two parts: *(1)* the initial Intra-EC Virtual Network Mapping, which provides the (partial or not) mapping of the VNE request within the ECs of the Edge Network and guarantees the constraint (4c), and *(2)* the main shortest paths-based algorithm that undertakes the composition of a DVNE solution, based on the initial VN mapping, with minimized round-trip delay, while dealing with the exponential complexity of the problem, achieving a fast performance in terms of execution time.

5. Delay-Aware Distributed VNE Solution

5.1. Initial Intra-EC VN Mapping

For the initial VN mapping, various existing works rely on heuristic approaches to deal with the combinatorial nature of the problem, as the VNE is an *NP-Complete* problem [28]. Typical approximation algorithms for VNE concern *bin-packing problem* approaches, like *First-Fit algorithm*, for the initial VNF mapping of a VN within an EC infrastructure [29]. In this section, we propose a heuristic approach to undertake the initial mapping of the VNE request within the ECs. This algorithm associates every VNF of the VN with a set of candidates EC that can be hosted onto. In detail, the VNE request is processed for each EC, and the mapping that occurs for every VNF of the VN would be accepted by the corresponding EC, during the next phase of the distributed solution construction. It is worth mentioning, that a VN could be mapped onto an EC partially, depending on resource availability and other constraints, as these are described below. Aiming to minimize the servers' utilization and the bandwidth consumption within an EC infrastructure, while achieving a notable VNF pair co-location ratio, our approach leverages the fact that a VNE request includes predeployed shared-VNFs on specific servers of an EC, in order to provide a mapping solution.

We assume that the set $\Xi = \{\xi_1, \ldots, \xi_K\}$ contains all the active servers of an EC infrastructure. A server $\xi \in \Xi$ has an available computing capacity c_ξ. Furthermore, the physical links between the servers via a switch are defined by the set $\mathcal{L} = \{l_\xi \mid \xi \in \Xi\}$ and every link is associated with a certain available bandwidth value β_{l_ξ}. As we already mentioned, the shared-VNFs are already deployed and hosted on specific EC servers. We assume that the set $SH = \{s_i | s_i \in V_s : rs_i = 0\}_i$, contains the shared-VNFs of the request, where $i = 1, \ldots, \mu$. We also define the function $h(s, \xi)$, to express whether a server ξ hosts the VNF $s \in V_s$,

$$h(s, \xi) = \begin{cases} 1, & \text{if } \xi \text{ is the host of } s \\ 0, & \text{otherwise} \end{cases}. \quad (5)$$

The mapping of a VNF s_i in a server ξ has to satisfy the following constraints regarding the computing capacity demand rs_i and the bandwidth demand $b_{i,i+1}$, between adjacent VNFs.

(1) Computing capacity constraint: This constraint inspects the mapping feasibility based on the aforementioned policy. Particularly, for shared-VNFs examines if the candidate server ξ is the corresponding host, while for VNFs to be deployed scrutinizes the available capacity in the server with respect to the VNF demand. These define the following parameter α_1:

$$\alpha_1 = \begin{cases} 1, & \text{if } h(s_1, \xi) = 1 \;\&\; s_i \in SH \text{ (deployed in } \xi) \\ 1, & \text{if } rs_i \leq c_\xi \;\&\; s_i \notin SH \text{ (to be deployed in } \xi) \\ 0, & \text{otherwise} \end{cases}. \quad (6)$$

Algorithm 1 Intra-EC VN Mapping Heuristic

Input: G_s, $\mathcal{N}_v$, Ξ, $\mathcal{L}$, SH
Output: A mapping $\mathcal{H}$ of VNFs of a VNE request

1: $\mathcal{U} \leftarrow \{V_s^1, \ldots, V_s^\mu\}$
2: **for** $V_s^i \in \mathcal{U}$ **do**
3: $\quad \Xi' \leftarrow$ sort Ξ based on c_ξ in reverse order
4: $\quad$ **if** $i > 0$ **then**
5: $\quad\quad$ Bring host of $s_{i-1} \in SH$ on top of Ξ'
6: $\quad$ **end if**
7: $\quad$ mapped $\leftarrow$ PARTITIONMAPPING(V_s^i, Ξ', $\mathcal{L}$)
8: $\quad$ **if** not mapped **then**
9: $\quad\quad$ re-partition V_s^i based on resource requirements
10: $\quad\quad$ mapped $\leftarrow$ REPARTITIONMAPPING(V_s^i, Ξ', $\mathcal{L}$, 1)
11: $\quad$ **end if**
12: **end for**
13: **procedure** PARTITIONMAPPING(V_s^i, Ξ', $\mathcal{L}$)
14: $\quad$ Bring host of $s_i \in SH$ on top of Ξ' if $s_i \in \mathcal{N}_v$
15: $\quad$ Reverse the order of the V_s^i
16: $\quad$ **for** $s_j \in V_s^i$ **do**
17: $\quad\quad$ check $\leftarrow$ *False*
18: $\quad\quad$ **for** $\xi \in \Xi'$ **do**
19: $\quad\quad\quad$ **if** (6) and (7) satisfied **then**
20: $\quad\quad\quad\quad$ check $\leftarrow$ *True*
21: $\quad\quad\quad\quad$ Map s_j on server ξ
22: $\quad\quad\quad\quad$ Update values of c_ξ and β_{l_ξ}
23: $\quad\quad\quad\quad$ Bring ξ on top of Ξ'
24: $\quad\quad\quad$ **end if**
25: $\quad\quad$ **end for**
26: $\quad\quad$ **if** not check **then**
27: $\quad\quad\quad$ **return** *False*
28: $\quad\quad$ **end if**
29: $\quad$ **end for**
30: $\quad$ **return** *True*
31: **end procedure**
32: **procedure** REPARTITIONMAPPING(V_s^i, Ξ', $\mathcal{L}$, δ)
33: $\quad$ **if** $\delta = |V_s^i|$ **then**
34: $\quad\quad$ **return** *False*
35: $\quad$ **end if**
36: $\quad$ *newParts* $\leftarrow |V_s^i| - \delta$ length parts of V_s^i
37: $\quad$ Sort *newParts* in descending order based on resource demands
38: $\quad$ **for** $V_s^{i*} \in$ *newParts* **do**
39: $\quad\quad$ **if** PARTITIONMAPPING(V_s^{i*}, Ξ', $\mathcal{L}$) **then**
40: $\quad\quad\quad$ **return** *True*
41: $\quad\quad$ **end if**
42: $\quad$ **end for**
43: $\quad \delta \leftarrow \delta + 1$
44: $\quad$ REPARTITIONMAPPING(V_s^i, Ξ', $\mathcal{L}$, δ)
45: **end procedure**

(2) Bandwidth capacity constraint: The bandwidth of the virtual link (s_i, s_{i+1}), $b_{i,i+1}$ is allocated by the mapping of the VNF s_{i+1}, as for each partition the mapping sequence is reversed. Although, if the s_i is co-located with the s_{i+1}, the previous bandwidth demand is released, and the next virtual link demand is allocated, which is $b_{i-1,i}$. Thus, the following parameter is defined to express the bandwidth constraint of a VNF s_i mapping in the candidate server ξ:

$$\alpha_2 = \begin{cases} 1, & \text{if } h(s_{i+1}, \xi)=1 \text{ (same hosts)} \\ 1, & \text{if } b_{i,i+1} \leq \beta_{l_\xi} \;\&\; h(s_{i+1}, \xi)=0 \text{ (different hosts)} \\ 0, & \text{otherwise} \end{cases} . \quad (7)$$

where, for s_i to be mapped onto server $\xi \in \Xi$, if $h(s_{i-1}, \xi) = 0$, the demanded bandwidth of the virtual link has to be available in the physical link l_ξ, while in the case where $h(s_{i-1}, \xi) = 1$, which means that the adjacent VNFs are co-located, no bandwidth is consumed from the corresponding physical link l_ξ.

Algorithm 1 includes the steps toward the Intra-EC VN mapping. Initially, between the lines 1 and 12 the VN partition phase takes place. The partitioning of the VNE request is performed, based on the position of the shared-VNFs in the VN,

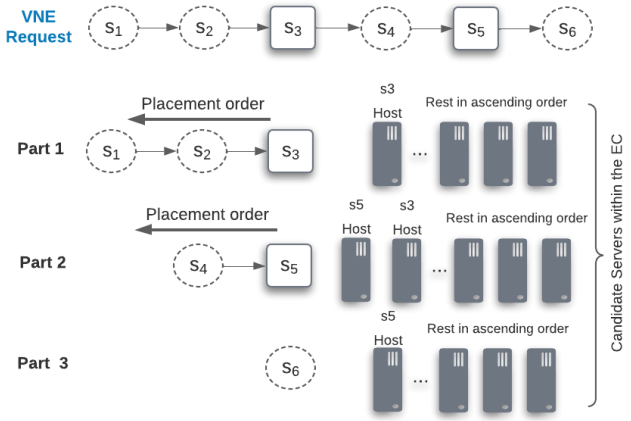


Figure 5: The proposed heuristic for Intra-EC VN Mapping.

where each partition is a path of VNFs, starting from a new VNF for deployment and reaches the shared-VNF as a destination on the partition path (line 1). Assuming μ shared-VNFs in V_s, we denote $\mathcal{U} = \{V_s^1, \ldots, V_s^\mu\}$ as the set that contains the partitions of V_s, where $V_s^1 \cup V_s^2 \cup \ldots \cup V_s^\mu = V_s$. Each partition mapping begins from the host server of the shared-VNF, while the rest available servers are sorted in ascending order (Ξ') on their available resources (line 3). Between the partitions of the same VN, a *Best-Fit* policy is adopted, meaning that the next partition that has to be mapped will start its VNF mapping from its corresponding shared-VNF (line 5), while next on the list of candidates servers will follow the hosts of the previous partition. The mapping of a partition is undertaken by the PARTITIONMAPPING() procedure (line 7). If a partition is not able to be mapped due to unavailable resources within the EC, it will be further partitioned according to its resource demands (line 9), and the mapping procedure will be invoked again (line 10). The mapping of a VN partition is performed by the PARTITIONMAPPING() procedure (lines 13-31). The mapping process begins from the host server of the shared-VNF, if the latter is hosted on this EC, and following the partition path backward (lines 14-15), the remaining VNFs are placed in a *Worst-Fit* fashion based on their resource demands (lines 16 - 29). In line 19, the mapping constraints are checked for each VNF of the partition. For enhanced VNF pair co-location and reduced inter-server network traffic, at each iteration, the host of the last placed VNF, emerges at the top of the list of candidates to host the next VNF (lines 20 - 23). The *check* parameter identifies when a partition is not able to fit within the EC, while it returns *False* (lines 26-27) as a result of the PARTITIONMAPPING() procedure and invokes the re-partitioning. If a partition of length $|V_s|$ does not fit in the EC servers, the heuristic selects sequences of VNFs of length $|V_s| - 1$ and maps one of them, starting from the one with the maximum resource requirements. If none of the $|V_s| - 1$ length partitions is mapped, the sequence of VNFs of length $|V_s| - 2$ is considered and so on. The process continues until no VNF remains after re-partitioning. The re-partition policy aims to maximize the number of adjacent VNFs that can be co-located during the mapping. The parameter δ determines

the number of sequential VNFs to be considered for the $|V_s| - \delta$ partitions. These partitions are sorted in descending order based on the total resource requirements. Then, the algorithm iterates through these partitions and maps the first that fits within the servers Ξ' by invoking the basic PARTITIONMAPPING() procedure.

Figure 5 illustrates an example, where a VNE request consists of two shared-VNFs from a total of six. Following the above described policy, the VN is partitioned based on the position of the shared-VNFs in three distinct partitions. For the sake of demonstration, the mapping process for the first partition $V_s^1 = \{s_1, s_2, s_3\}$ is performed as follows: The mapping attempt starts backwards from the VNF s_3. As s_3 is a shared-VNF, it is already placed in a server, which is its host, denoted by $\xi_a \in \Xi$, and $h(s_3, \xi_a) = 1$. The ξ_a is now the first candidate in the list of servers Ξ' for the VNF s_2 to be mapped, based on resource constraints, aiming to minimize the utilized servers and the inter-server network traffic; otherwise, the *Worst-Fit* is performed. Then, for the s_1 VNF, the first candidate server in the list Ξ' would be the host of s_2.

The output of the algorithm 1 is a mapping $\mathcal{H} = \{\eta_i\}_i$, where $i = 1, \ldots, m$. In specific, η_i denotes a server $\xi \in \Xi$ that hosts the VNF $s_i \in V_s$, while in the case where the VNF s_i has no host in the EC, it stands that $\eta_i = None$. Thus, EC can be a candidate host for the VNFs that are mapped as occurs for $\mathcal{H}$. The mapping $\mathcal{H}$ will be used below, from the algorithm that undertakes the distributed embedding of the VN.

5.2. Distributed VNE with Minimum Round-Trip Delay

The methodology for solving the DVNE problem of Section IV-B is presented here. Towards the minimization of the round-trip delay, we implement a shortest path-based approach that relies on the Yen's *k-shortest-paths* algorithm [30], which calculates the k shortest loopless paths between two nodes in a graph (weighted or not), by utilizing the *Dijkstra's shortest-path algorithm*. As mentioned, in typical SFC embedding problems, a distinct VNF is the destination of the execution sequence. Imitating this approach, the proposed algorithm, namely *kSP-DVNE*, generates paths considering that every VNF $s \in V_s$ could be the destination of a VNP and for every potential destination, a forward VNP (f-VNP) and a backward VNP (b-VNP) are determined. For example, taking into consideration the VNE request in Fig. 3, assuming that the destination is the VNF s_2, the corresponding f-VNP is $\{s_0, s_1, s_2\}$, while the b-VNP is $\{s_2, s_3, s_4, s_0\}$. Additionally, a Virtual Network Embedding Path (VNEP) is defined for both the f-VNP and b-VNP, denoted as f-VNEP and b-VNEP respectively. These embedding paths determine the visited EC nodes that host the corresponding VNFs of f-VNP and b-VNP. The {f-VNEP} ∪ {b-VNEP} provides a DNVE solution $\mathcal{P}$, which is actually a circuit on the graph G, as an EC is able to be visited multiple times as it could host various VNFs of the VNE request. For the above example, the determined embedding paths are: f-VNEP= $\{u_0, u_1, u_2\}$ and b-VNEP= $\{u_2, u_3, u_4, u_0\}$. Below, a description of the main components of the proposed solution is given.

5.2.1. Augmented Graph Construction

The initial Intra-EC VN mapping solution provides the candidate ECs for hosting every VNF $s \in V_s$. This mapping is denoted by $\mathcal{H}_v$, $\forall v \in V$. The implementation of the proposed DVNE method relies on an *augmented graph* $\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a)$, where the graphs G and G_s that represent the edge network and the VNE request are merged. In specific, the set $\mathcal{V}_a = V \cup V_s$ represents the graph's vertices and consists of these that correspond to the EC nodes V plus the VNFs nodes V_s. Furthermore, the set $\mathcal{E}_a = E \cup \{(v, s, 0) \mid \forall v \in V \text{ and } \forall s \in V_s \cap \mathcal{H}_v\}$ represents the augmented graph's edges. The latter consists of the edges of the initial substrate network links E, while for every EC v that is a candidate host for VNF s, an edge of zero weight $(v, s, 0)$ is added. For example, we consider the substrate network graph model of Fig. 2 and the VNE request of Fig. 3. A possible augmented graph $\mathcal{G}_a$ is depicted in Fig. 6. In this example, we assume that, depending on the mapping of each EC, the VNF s_1 can be hosted $\{v_1, v_2, v_3, v_6\}$. So, $\mathcal{E}_a$ contains all the edges of the set E, while adding to it the edges $(v_1, s_1, 0), (v_2, s_1, 0), (v_3, s_1, 0)$ *and* $(v_6, s_1, 0)$. Similarly, edges from candidate ECs for hosting the remaining VNFs are accordingly added in the graph $\mathcal{G}_a$. The construction of the augmented graph is essential for the development of the proposed solution for the round-trip delay minimization. It is worth mentioning that the network delay between two co-located VNFs in the same EC is considered negligible and is determined equal to zero.

5.2.2. kSP-DVNE algorithm

Regarding DVNE, the proposed approach takes into account the characteristics of the typical SFC embedding problem. As mentioned above, a VNE request is submitted in a specific EC. The VNFs that comprise the VN, can be hosted in various ECs of the Edge Network. Some of the VNFs of the VNE request have to be embedded with respect to resource constraints, while the shared-VNFs are already deployed on specific ECs. Also, the execution outcome of the VN has to be directed back to the source; that is the end-device, which generated the request. This implies no specific destination EC for the network traffic, nor a predefined forward and backward Service Function Path. To deal with this abstract nature of the VN, our approach aims to examine various instances of the VN, with different destination VNF (and a corresponding host EC) following different paths toward the respective destination VNF per case. Obviously, a VNF s could have several candidates hosting ECs v, and various paths from the source to the destination EC v. The number of the possible different destinations VNFs of a VN equals to the VN's length, m. Consequently, m will also be the number of the distinct f-VNPs and b-VNPs. Aiming on providing the DVNE solution with minimized round-trip delay, the proposed approach is based on the following remark:

Remark 1. *Let $\mathcal{SP} = \{p_{sh}(s_1), p_{sh}(s_2), \ldots, p_{sh}(s_m)\}$ the shortest paths from a source v_0 to every VNF $s_i \in V_s$ of a VN in $\mathcal{G}_a$. If p_{sh}^{max} is the maximum cost path in $\mathcal{SP}$ and s_{dst}^{sh} the corresponding destination VNF, the minimum round-trip delay of a DVNE*

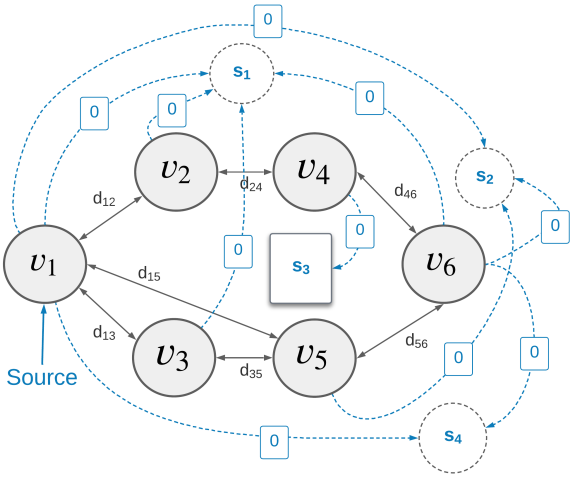


Figure 6: An augmented graph $\mathcal{G}_a$ based on G and G_s.

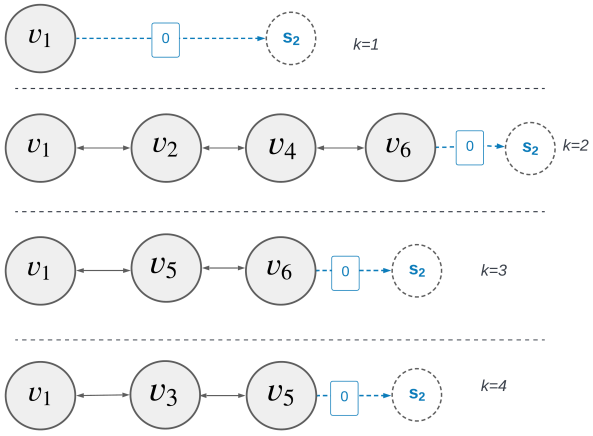


Figure 7: An example of four possible shortest paths from s_0 to s_2 in $\mathcal{G}_a$.

is:

$$D^*_{rt} \geq 2 \times w(v_0, s^{sh}_{dst}).$$

Given the fact that the network traffic has to traverse every host u_i of each VNF s_i, in the best case, the shortest reachable host for a feasible DVNE solution, is this that corresponds to p^{max}_{sh}. This path corresponds to a destination VNF s^{sh}_{dst}. If the f-VNP and respectively the b-VNP can be embedded in the p^{max}_{sh}, in a way that every edge is traversed at most once from the f-VNEP and b-VNEP individually, the round-trip delay D^*_{rt} of this embedding solution is the minimum achievable. Therefore, the main goal of this approach is to generate multiple paths that could be examined in a similar way; that is to find a f-VNEP and a b-VNEP on the same path, while the path's edges are traversed only once for the corresponding f-VNEP and b-VNEP. Below, more details regarding the proposed method are given.

Candidate Embedding Paths: Towards the generation of a sufficient number of paths for every destination VNF, we rely on the Yen's *k-shortest paths (k-SP)* algorithm [30], which provides the $k \in \mathbb{N}$ shortest loopless paths between two vertices in a graph. Given a VNE request of m VNFs and a source EC v_0, we consider the m possible destination VNFs $s_{dst} \in V_s$. The Candidate Embedding Paths are generated by computing k shortest paths from u_0 to each VNF s_{dst}. We define the k shortest paths between the source EC and a VNF s as:

$$P^k_s = \{p^1(s), p^2(s), \ldots, p^k(s)\}, \tag{8}$$

where $p^i(s)$ is the i^{th} shortest path from v_0 to s in $\mathcal{G}_a$ and contains the traversed ECs $v \in V$ with the sequence that they are visited. Thus, the candidate embedding paths constitute the set:

$$C = \bigcup_{s \in V_s} P^k_s. \tag{9}$$

As m is the number of different destination VNFs, also m distinct pairs of f-VNPs and b-VNPs are determined. Subsequently, the paths of C will be examined for efficient DVNE solutions, in terms of round-trip delay.

***k*SP-DVNE algorithm**: Algorithm 2 describes the steps to provide a solution for the DVNE problem. To begin with, the k shortest paths in the augmented graph $\mathcal{G}_a$ are computed, from the source EC v_0 to each vertex that corresponds to a VNF of the VN (lines 1-3), as described previously. Afterward, each candidate path $p_c \in C$ is examined (line 6) based on the following policy: A candidate path p_c must contain at least one host EC for every VNF $s \in V_s$. Given the augmented graph $\mathcal{G}_a$, a candidate path $p_c \in C$ is able to provide a DVNE solution if the following condition is satisfied:

$$\forall s \in V_s, \exists v \in p_c : \exists (v, s) = 0, \ where \ (v, s) \in \mathcal{E}_a. \tag{10}$$

In essence, an edge between an EC v and a VNF s must exist in $\mathcal{G}_a$, so, the path $p_c \in C$ is a *valid candidate* path.

f-VNP and b-VNP Embedding : Subsequently, for a valid candidate path p_c, an embedding solution is constructed. More precisely, as described above, the solution construction can be divided into the f-VNP and b-VNP embedding, which leads to the f-VNEP and b-VNEP, respectively (lines 7-11). Regarding the f-VNP = $\{s_1, \ldots, s_{dst}\}$, the process starts from the s_{dst} host, which is the last EC vertex of p_c. Accessing the f-VNP in reverse order, (*i.e.,* $s_{dst-1}, \ldots, s_1$), we seek for the first available host of the next VNF, following the p_c vertices, also, in the opposite direction. For the b-VNP case, we access the VNFs straight forward (*i.e.,* $s_{dst+1}, \ldots, s_m$), while we seek for the first available host of the next VNF, following the p_c vertices in the direction towards the source EC. The DVNE solution is defined based on the f-VNEP and b-VNEP, its round-trip delay is computed (lines 12-18). Figure 8 illustrates an example of the embedding solution construction, considering the path $p^2(s_2)$ of Fig. 7. The destination VNF s_2 is placed in the corresponding host EC of the candidate path v_6. The EC v_1 is the source EC, and so $v_0 = v_1$, while the f-VNP is $\{s_0, s_1, s_2\}$ and the b-VNP is $\{s_2, s_3, s_4, s_0\}$. Regarding the end-device, which is the source s_0 of the network traffic, the EC that it is connected is the $v_0 = v_1$. Starting the embedding from f-VNP, the s_2 is hosted on v_6. The next VNF to be placed is the s_1. The first EC that is a candidate host for s_1 from v_6 towards v_1 will be chosen. In this case, v_6

Algorithm 2 Delay Aware kSP-DVNE Algorithm

Input: $\mathcal{G}_a$, G, G_s
Output: $\mathcal{P}^*$, D^*_{rt}

for $s \in V_s$ **do**
  $P^k_s \leftarrow$ `k-Shortest Paths` from u_0 in $\mathcal{G}_a$
end for
$C \leftarrow \{P^k_s\}$, $\forall s \in V_s$
for $p_c \in C$ **do**
  if (10) is satisfied **then**
    Identify the destination VNF s_{dst} in p_c
    f-VNP $\leftarrow \{s_0, \ldots, s_{dst}\}$
    b-VNP $\leftarrow \{s_{dst}, \ldots, s_0\}$
    # Embed f-VNP on p_c (reverse order)
    f-VNEP $\leftarrow \{u_0, u_1, \ldots, u_{dst}\}$
    # Embed b-VNP on p_c
    b-VNEP $\leftarrow \{u_{dst}, \ldots, u_m, u_0\}$
    Construct the embedding solution
    $\mathcal{P} \leftarrow$f-VNEP $\cup$ b-VNEP
    if (4b) satisfied **then**
      $D_{rt} \leftarrow \sum_{i=0}^{m} w(u_i, u_{i+1})$, $u_i \in \mathcal{P}$
      if (4d) satisfied **then**
        Check if $D_{rt} < D^*_{rt}$
        Update D^*_{rt} and $\mathcal{P}^*$ if needed
      end if
    end if
  end if
end for

is nominated as a host of s_1. The f-VNEP is then $\{v_1, v_6, v_6\}$, containing the host ECs for the corresponding services. Concerning the b-VNP embedding, the shared VNF s_3 has only one candidate host EC v_4, where it is already placed. Then, for the s_4, exclusively the ECs after the v_4 and backward (toward v_0) are taken into account for the identification of a candidate host, where v_1 is the only choice for s_4. In this way, the b-VNEP is shaped as $\{v_6, v_4, v_1, v_1\}$. The construction of the final solution is the circuit starting from the $v_0 = v_1$ and traversing the respective hosts of each VNF of the request, observing the execution order in which they are defined. Thereafter, the DVNE solution is,

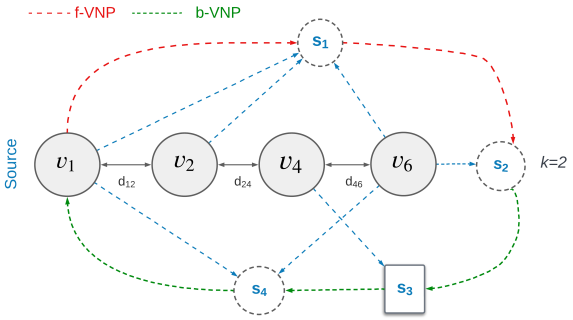
$$\mathcal{P} = \{u_0, u_1, \ldots, u_4, u_5\} = \{v_1, v_6, v_6, v_4, v_1, v_1\},$$

as $v_0 = v_1$. The round-trip delay of this solution is calculated using the shortest paths between the traversed ECs and equals to:

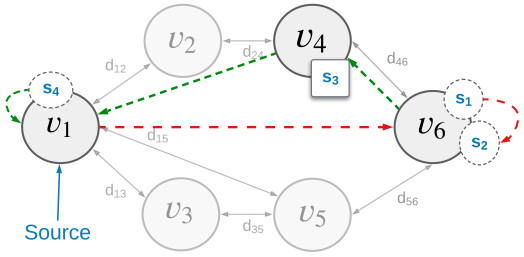
$$D_{rt}(\mathcal{P}) = \sum_{i=0}^{4} w(u_i, u_{i+1}).$$

5.2.3. Complexity Analysis

The proposed DVNE solution can be divided into three major parts: (i) the augmented graph $\mathcal{G}_a$ construction, (ii) the k-shortest paths set $C = \{P_k\}$ calculation and (iii) the computation of the solution with the lowest round-trip delay $D_{rt}, \mathcal{P}$. From this perspective, we can analyze the complexity of the solution as follows:



(a) A candidate embedding path of a VN with destination VNF s_2.



(b) DVNE solution provided by the proposed kSP-DVNE heuristic.

Figure 8: A DVNE solution in the $p^2(s_2)$ path of Fig. 7.

(i) The construction of $\mathcal{G}_a = (\mathcal{V}, \mathcal{E})$ is based on $G = (V, E)$ and $G_s = (V_s, E_s)$. Every vertex $v \in V$ is examined as a candidate host for a subset of the VNFs $s_i \subset V_s$. So, for each $v \in V$, we iterate the VNFs which belong to the mapping $\mathcal{H}_v$ and the corresponding edges of zero weight are added in the graph accordingly. Let $n = |V|$ and $m = |V_s|$, the augmented graph construction demands $n \times |\mathcal{H}_v|$ operations and, as $|\mathcal{H}_v| \leq |V_s| = m$, $\forall v \in V$, (at most m VNFs can be hosted in an EC), the time complexity is $O(nm)$.

(ii) According to Algorithm 2, the following step is to compute the k-shortest paths, in order to create the set of the candidate paths to be examined. To achieve this, we utilize Yen's algorithm in the graph $\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a)$. As it described in [30], the time complexity of the algorithm $O(k|\mathcal{V}_a|^3)$. With respect to our modeling, it is valid that $\mathcal{V}_a = V \cup V_s$, therefore, $|\mathcal{V}_a| = |V| + |V_s| = n + m$. Therefore, the time complexity of that algorithmic step is $O(k(n+m)^3)$.

(iii) In order to provide the solution, the shortest path cost w that represents the network delay between every pair of nodes in V has to be calculated in prior for each VNE request. As described above, the round-trip delay of an embedding solution is $D_{rt} = \sum_{i=0}^{m} w(u_i, u_{i+1})$. Using the Floyd-Warshall all pair shortest paths algorithm [31], we obtain the desired results in the complexity of $O(n^3)$.

(iv) The candidate paths list is accessed just once in order to find the DVNE solution, and its length is equal to km. In an iteration, each path is validated if contains hosts for all services. At most, a path consists of n vertices that will have to be examined, as we care about vertices that represent ECs. Furthermore,

two more times, the path is traversed for providing the mapping of the f-VNP and b-VNP, thus, $2n$ at most vertices could be accessed. The round-trip delay calculation is the sum of the pairs of vertices in $\mathcal{P}$, which have cardinality equal to $m + 2$. Hence, the time complexity of the loop that calculates the DVNE solution is:

$$O(km(n + 2n + (m + 2))) = O(km(n + m)).$$

In summary, the complexity of the proposed algorithm is:

$$O(nm + k(n + m)^3 + n^3 + km(n + m)) = O(kn^3), \qquad (11)$$

as $m < n$, where $m, n \in \mathbb{N}$. It is obvious that the algorithm's complexity depends on the k value, which determines the number of the candidate paths for the DVNE. Higher value of k implies larger search space for the algorithm and increased chances for finding a near optimal (or the optimal) embedding solution. Additional remarks regarding the effect of the k parameter on the complexity of the algorithm are presented in the evaluation section. At next, we present a performance evaluation of both the initial Intra-EC placement Heuristic and the kSP-DVNE algorithm through simulation and comparison with relative studies.

6. Evaluation

In this section, the evaluation of the proposed approach that deals with the DVNE problem is presented in two distinct parts. Initially, the heuristic Intra-EC VN mapping performance is analyzed and compared with a relative approach in the literature [29]. Subsequently, the Distributed VNE solution is evaluated and furthermore compared with an existing work regarding delay-aware hybrid-SFC embedding in a distributed manner at the network edge [12]. For every experiment, the impact of major parameters on the DVNE solution is illustrated through numerical results and relevant discussions.

We implemented both sV-VNM and kSP-DVNE approaches, as well as the overall simulation environment using Python programming language. The same stands for the relevant approaches from the literature that are utilized to perform the comparisons with the algorithms that we propose in this paper. The ECs and the VN are implemented as two separated objects containing the computing and network attributes of the aforementioned system modeling. To generate the graph based models of the Edge Network and the VNE request, the Python package *NetworkX* [32] is utilized. This package enables the corresponding parameters of the model to be parsed on the package's graph objects, as nodes' and links' attributes. The simulations took place on a Virtual Machine with installed operating system *Ubuntu 20.04*, with 4 virtual CPU cores and 16GB of RAM.

6.1. Intra-EC VN Mapping Heuristic Evaluation

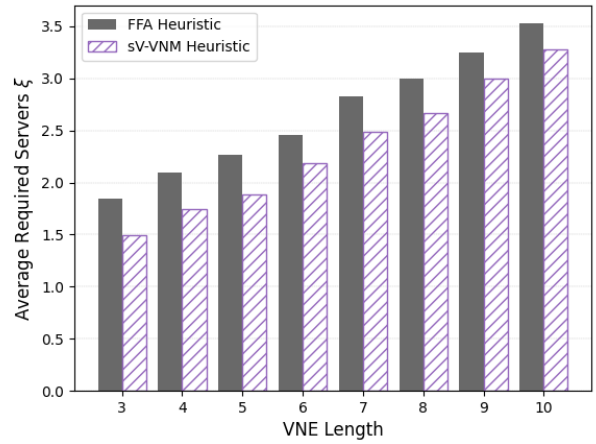
Regarding the proposed heuristic for the Intra-EC VN mapping, namely *shared VNF-based VN Mapping Heuristic (sV-VNM)*, the evaluation is performed through modeling and simulation, while as mentioned before a comparison with a similar approach [29] is also provided. This approach, namely First-Fit Allocation heuristic (FFAh) allocates the VNFs of an SFC in a group of servers, which operate under the same core switch in an Edge Network. Similarly to sV-VNM, the FFAh aims at providing an initial VNF mapping towards the minimization of the power consumption (i.e., number of activated servers) within the EC and the minimization of the aggregated network traffic that the VNFs inject between the servers. Furthermore, this work leverages the VNF co-location to achieve its objectives. The main idea of the FFAh can be described as follows. For every cluster of servers, which compose an EC in our case, the FFAh sorts the servers in decreasing order with respect to their available resources. Afterward, the FFAh strives to allocate each VNF of the SFC in a first-fit fashion. The above remarks make the FFAh a suitable alternative for comparison with the proposed sV-VNM approach.

Table 3: sV-VNM Evaluation - Simulation Parameters.

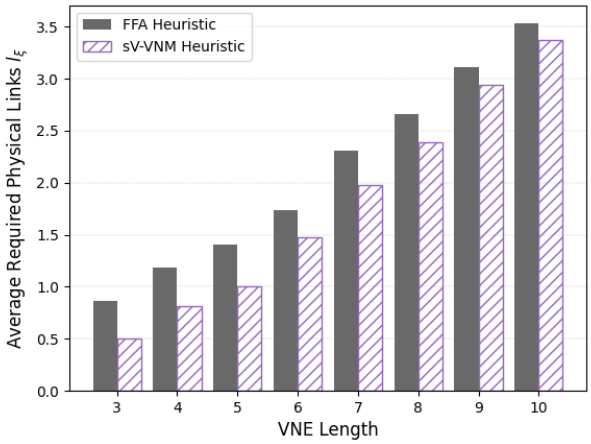
Parameter	**Value**
Number of ECs $\|V\|$	100
Number of active servers $\|\Xi\|$ per EC	$U[5, 10]$
VNE Length $\|V_s\|$ per request	$U[3, 10]$
Shared-VNFs in every request	$20\% - 30\%$ of $\|V_s\|$
Available cores per server	$U[2, 16]$
Available bandwidth per Intra-EC link	1 Gbps
Demanded cores per VNF	$U[1, 5]$
Demanded Bandwidth per virtual link	$U[30, 150]$ Mbps
Total VNE requests	1000

6.1.1. Simulation Setup

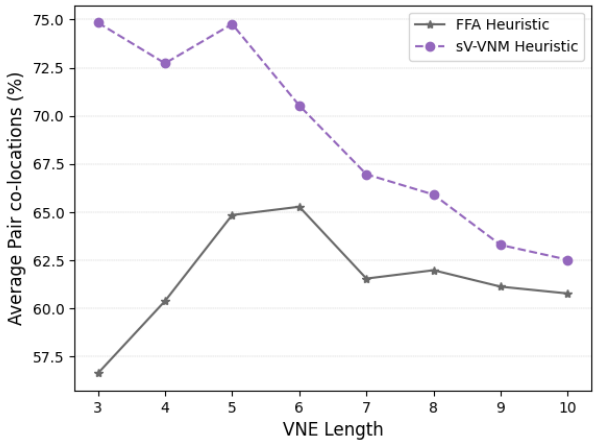
With respect to our system modeling, the EC is the corresponding cloud data center and it contains a group of available servers linked with the corresponding switch of the EC. The number of ECs is equal to 100, while the number of active servers $\Xi = \{\xi_i\}_i$ within each of them varies in the discrete uniform space $[5, 10]$. Regarding the computing and network resources, the available cores for each server, c_ξ, ranges from 4 to 16, while the available bandwidth capacity β_{l_ξ} of each l_ξ link, is set to 1 Gbps. Furthermore, we consider a pool of 20 distinct type of VNFs that a subset can be selected from, in order to compose a VNE request. Concerning the VNE request, its length ranges from 3 to 10, while up to 3 of these VNFs are shared-VNFs. For the rest VNFs, the demanded CPU cores are uniformly distributed in the space $[1, 5]$. The bandwidth demand of the virtual links, between adjacent VNFs ranges from 30 to 150 Mbps. In order to ensure fairness in comparative results regarding the performance of the considered heuristics, we assume that all the demanded shared VNFs of each VNE request are available in any of the ECs during the experiment. Each VNE request is submitted in any of the ECs of the substrate network, which, as mentioned above, will have different resource availability, and different placement of the shared VNFs. In total, 1000 VNE requests are generated and forwarded to all of the ECs. Table 3 includes all simulation parameters of the Intra-EC VN Mapping experiment. The comparison



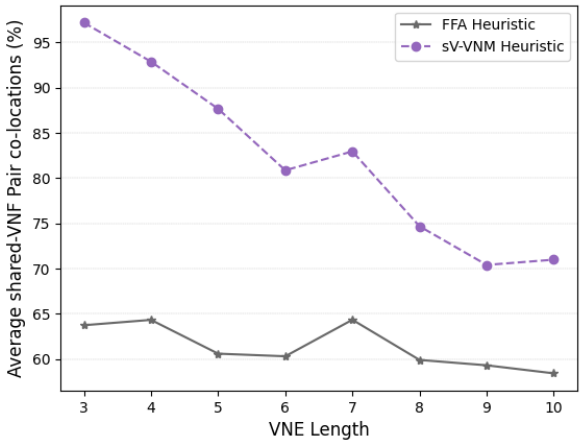
(a) Average number of utilized servers per EC.



(b) Average number of utilized links per EC.



(c) The pair co-location ratio between adjacent VNFs.



(d) The pair co-location ratio of adjacent VNF and shared-VNF.

Figure 9: Experiment Results - Comparison between the sV-VNM Heuristic and FFA Heuristic.

of the two heuristics is based on the following metrics: (i) The average number of servers used per VNE request in every EC, (ii) the average number of physical links that are utilized (injected traffic between servers) in every EC, (iii) the percentage of the co-located on the same server adjacent VNFs and (iv) the percentage of pair co-location involving shared-VNFs with respect to the length of the VNE request.

6.1.2. sV-VNM and FFAh Comparative Results

The numerical results are illustrated in Fig. 9. Regarding the average number of used servers, the sV-VNM utilizes 12-20% less servers regarding VNE of small and medium lengths (3-6 VNFs in the VNE) compared to the FFAh, while it still performs slightly better for larger VNE requests (7-10) with 6-10% less utilized servers, as it is shown in Fig. 9a. Similar results occurred regarding the physical links allocated for every request. On average, the sV-VNM utilized approximately 18% less physical links to allocate the virtual links of a VNE (Fig. 9b). The performance of the proposed sV-VNM is better in both aforementioned metrics, especially in the small and medium length VNEs, where the percentage of the shared-VNFs is higher, compared with the larger requests that the two heuristics perform similarly. Additional results concern the pairing ratio of adjacent VNFs that the heuristics achieve.

Figures 9c and 9d demonstrate the co-location ratio between adjacent VNFs in general and those that include shared-VNFs respectively. Co-location ratio determines the efficiency of the VNFs mapping with regard to the servers' and links' utilization. Specifically, the co-location of adjacent VNFs leads to reduced network traffic between different servers and limited network delay between the corresponding VNFs, which is also beneficial regarding the QoS of a VN, as also stated in [33]. The proposed sV-VNM heuristic accomplishes an average co-location ratio of 69.8% in the general case, which is a noticeable improvement compared to the 62.7% of the FFAh. As for the pairs that involve shared-VNFs the corresponding ratio reached by the sV-VNM ranges from 95% for small length VNE requests to a minimum of 68% for larger VNE lengths. On the other hand, FFAh provides an average 60% co-location

ratio of pairs that include shared-VNFs, which is approximately 20% less than sV-VNM. The results reflect the efficiency of the mixed policy adopted by the proposed algorithm. In particular, the partitioning based on the shared-VNFs and the placement of them in a Best-Fit fashion leads to a higher co-location ratio between the adjacent VNFs that connect the distinct VNE partitions. As mentioned above, each partition mapping starts from the host of the respective shared-VNF (if exists), while the worst-fit policy is used when the least recently used server stays on top of the candidate list for mapping the next VNF. This VNE partition mapping policy gives an advantage to the sV-VNM compared to the FFA heuristic. Nevertheless, for a single VNE partition mapping, which does not include shared-VNF, the two heuristics are expected to perform equally.

6.2. *kSP-DVNE Algorithm Evaluation*

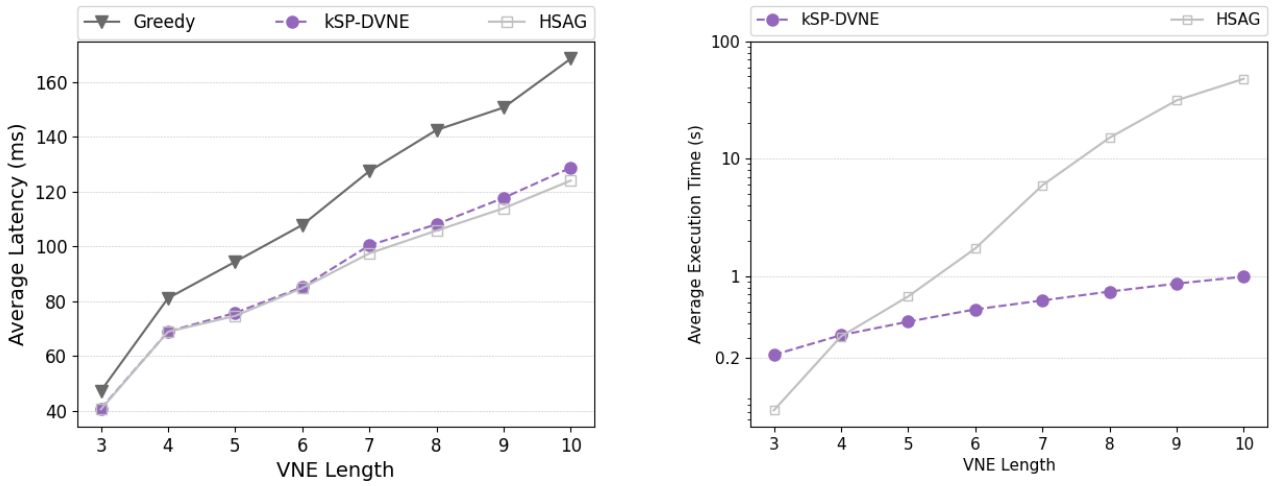
We hereby present the performance evaluation of *k*SP-DVNE algorithm. The proposed solution is compared with a technique introduced in [12], which tackles the issue of minimizing the network latency during hybrid-SFC embedding in an edge network environment. The authors in [12] design their solution on the Hybrid-SFC Embedding Auxiliary Graph (HSAG), the main components of which are defined as: (i) the *tier* that corresponds to the VNF (or Service Functions - SFs) of the h-SFC, (ii) the *layer* of the substrate candidate EC nodes of every VNF (or SF) and (iii) the *nodes*, which reflect to the substrate EC nodes in our case. Each tier contains a set of layers, where each layer is composed of nodes, which correspond to the ECs that are candidates to host the related SF. A solution is constructed by examining all the possible paths that occur via the auxiliary graph. When the nodes of a layer are examined, for every one of them a new layer is defined in the next tier. Thus, starting from the initial tier, which includes only the source node, the paths to the candidate nodes for hosting the next VNF (or SF) are calculated. Then, for each tier's nodes, the shortest paths to the next VNF in the sequence of the h-SFC are calculated, until reaching the source node again. Consequently, we refer to the solution of [12] as HSAG, which achieves the minimal network latency in the h-SFC embedding problem. Despite the fact that it is a faster solution than the brute force, it is still of high computational complexity in the worst case, in which the VNFs have a large number of candidate host nodes. This is due to the fact that the number of layers on each tier grows faster, while the same stands for the alternative paths, which are examined in the solution path construction. The discussed DVNE problem is equivalent to the h-SFC problem with no-specific destination substrate node, as presented in [12]. We adapted the HSAG solution to our DVNE problem, and still it provides the solution with the minimal round-trip delay. However, our proposed algorithm achieves the optimal or near optimal solution much faster, making it suitable for IoT scenarios, not only for the initial embedding, but for different time sensitive network reconfiguration scenarios, where the VNE path has to be redefined in the substrate network. Furthermore, we implemented a greedy approach in addition to the proposed *k*SP-DVNE and the HSAG. The greedy algorithm is also a path-based approach, while it utilizes the Dijkstra algorithm to select in a greedy fashion, the next closest to the current substrate node, which is a candidate for the embedding of the examined VNF. In the augmented graph $\mathcal{G}_a$, starting from the source node v_0 the shortest path to the first VNF s_1 is calculated. The penultimate node of the path is considered as the host u_1 of the s_1 VNF. Consequently, the shortest path from u_1 to s_2 is computed, and so on, until the construction of a complete embedding solution. The greedy algorithm is used on our *k*SP-DVNE approach, as a lower bound solution of round-trip delay minimization for the DVNE problem, to deal with some cases that might occur, where no valid candidate paths are identified in the augmented graph. Regarding the greedy algorithm's complexity, if m is the length of a VNE request, m Dijkstra shortest paths are calculated, so the time complexity is $O(m(|V| + |E|)log|V|)$.

Table 4: *k*SP-DVNE Evaluation - Simulation Parameters.

Parameter	**Value**
Number of ECs (substrate nodes) $\|V\|$	40
VNE Length $\|V_s\|$ per request	$U[3, 10]$
Shared-VNFs in every request	$U[20\%, 30\%]\ of\ \|V_s\|$
Physical Link (v_i, v_j) delay (ms)	$U[5, 15]$
Available cores per EC server	$U[2, 16]$
Available bandwidth per Intra-EC link	1 Gbps
Demanded cores per VNF	$U[1, 5]$
Demanded Bandwidth per (s_i, s_j) link	$U[30, 150]$ Mbps
Total requests	700

6.2.1. *Simulation Setup*

The alternative algorithms for solving the DVNE problem were implemented using the Python programming language. Relying on real network topologies [34], we simulated an edge network composed of 40 EC infrastructures. A request for VNE can be submitted to any of these ECs, which length fluctuates uniformly from 3 to 10 VNFs, with the 20% to 30% of these being shared-VNFs on specific ECs. These shared-VNFs are derived from a pool of 20 in total. The computing resources demand for every VNF ranges uniformly for 1 to 5 CPU cores. Regarding the links between the ECs, the network delay varies from 5 to 15 ms. For each VNE length, 100 requests are generated. Based on this configuration, two different experiments are conducted. At first, we measure the efficiency of the proposed heuristic in comparison with the HSAG and the greedy solution, under a random mapping of the VNFs that compose a VNE request, following as much as possible a similar setting with [12]. Specifically, each EC can host between 2 and 3 out of the 20 available shared-VNFs. At the same time, an EC can be a candidate host of the 10% to 40% of the VNFs that compose a VNE request, following a uniform distribution. The average degree of the graph that represents the Edge Network is equal to 3, while the *k*SP-DVNE algorithm is configured to operate with the value of k equal to $40 \times 3 = 120$, that is the number of the EC nodes $|\mathcal{V}_a|$ multiplied by the average degree of the graph $\deg(G)$. More details regarding the selection of the k value are provided in subsection 6.3. Table 4 includes all key simulation



(a) Average Round-Trip Delay.

(b) Average Execution Time.

Figure 10: *k*SP-DVNE and HSAG comparison under random initial mapping of VNFs.

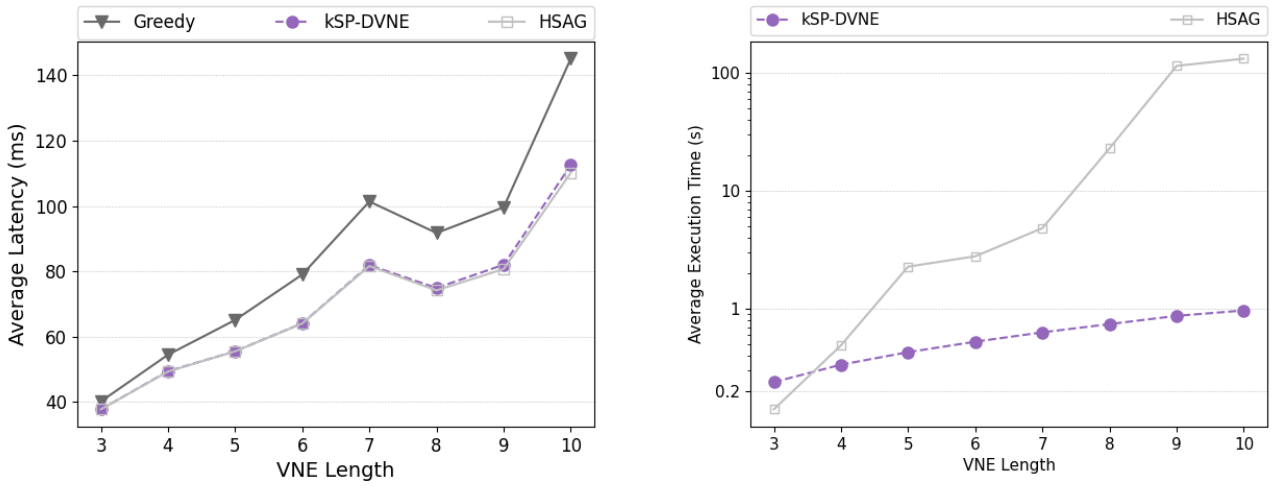
parameters for the evaluation of *k*SP-DVNE.

The second experiment aims at analyzing the effectiveness of the proposed *k*SP-DVNE combined with the initial Intra-EC heuristic. In detail, following the same experimental setup as above regarding the edge network parameters, as summarized in Table 4, the sV-VNM heuristic undertakes the initial mapping of the VN within the ECs. The Intra-EC parameters are, also, identical with the Intra-EC simulation parameters. In particular, these parameters are the *available cores per server* and the *available bandwidth per link*. During this experiment, the shared-VNFs availability is not constant among the ECs. Indicatively, per VNE request, the available shared-VNFs among the ECs are uniformly redistributed, in order to include a dynamic aspect in the simulations; that is to avoid the same shared VNFs to be always available in the same EC for all the VNE requests during the whole experiment. The number of shared-VNFs that each EC can provide for each VNE request varies from 2 to 3. Subsequently, the performance of the various compared DVNE approaches is analyzed, again under the mapping derived by the proposed sV-VNM heuristic.

6.2.2. *kSP-DVNE Evaluation Results*

Figure 10 shows the experimental results when the initial mapping of the VNFs s_i of a VNE request is performed randomly among the ECs. Both *k*SP-DVNE and HSAG outperform the greedy algorithm in terms of the round-trip delay minimization. The proposed kSP-DNVE achieves the minimum round-trip delay at 81% of the total requests. Compared to the HSAG, which provides the distributed embedding with the minimum round-trip delay, the *k*SP-DVNE produces embedding solutions with a slight increase of 1.88% in the average round-trip delay, for all the VNE lengths. Figure 10a depicts the round-trip delay of the embedding derived by each algorithm with respect to the VNE length. Concerning the execution time, which is illustrated in Fig. 10b, the proposed *k*SP-DVNE outperforms the HSAG by providing the embedding of the VNE much faster due to its lower complexity. Specifically, the execution time of the *k*SP-DVNE increases linearly with respect to the VNE length, as it is explained in the complexity analysis above, reaching an execution time reduction of over 90% on average compared to the HSAG, especially for a higher number of VNFs in the VN. It is worth mentioning that in the execution time of the *k*SP-DVNE heuristic, also the running time of the greedy algorithm is included, since it is used to provide a lower bound of round-trip delay for our solution. Regarding the second experiment, where the initial mapping of the VNFs that compose a VN is performed using the proposed sV-VNM heuristic, the results are shown in Fig. 11. Concerning the comparison with the greedy algorithm, again, both kSP-DNVE and HSAG achieve better results in terms of the round-trip delay, as it is shown in Fig. 11a. In fact, the *k*SP-DVNE performance shows significant improvement, as in the 95% of the cases provides the solution with the minimum round-trip delay, while the deviation from the HSAG solutions this time is approximately only 0.7%. Similar results with the first experiment occurred regarding the execution time, as illustrated in Fig. 11b, where, still, the *k*SP-DVNE outperforms the HSAG. The fact that the initial mapping comes from the sV-VNM, which reflects to a higher co-location of adjacent VNFs, enables the improvement of the *k*SP-DVNE performance in terms of the round-trip delay minimization. The increase of the execution time of the HSAG, here, comes from the fact that the sV-VNM provides more candidate solutions compared to the previous experiment. Again, the execution time of the greedy algorithm is included in the corresponding *k*SP-DVNE heuristic.

In order to explain the reason behind that improvement, firstly, we discuss about the proposed heuristic weaknesses. The main shortcoming of *k*SP-DVNE is that during the examination of the candidate paths for embedding, it always chooses the next embedding node towards the destination VNF, during the embedding of the f-VNP, or towards the source node v_0 for the embedding of the b-VNP, respectively. Thus, during the



(a) Average Round-Trip Delay.

(b) Average Execution Time.

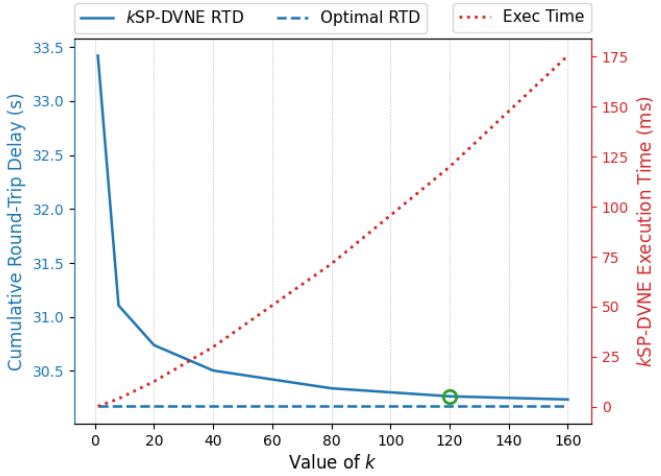
Figure 11: Comparison of the *k*SP-DVNE versus HSAG under initial mapping using sV-VNM.

embedding of the f-VNP or b-VNP, if there is a network path of low network delay between two nodes that have to be traversed several times (back and forth) to provide the optimal solution, it will not be chosen by the *k*SP-DVNE heuristic, unless it is the only solution in the current candidate path. Therefore, by increasing the co-location of the adjacent VNFs of a VN, the need for multiple traversals of the same network links to provide the optimal results is eliminated.

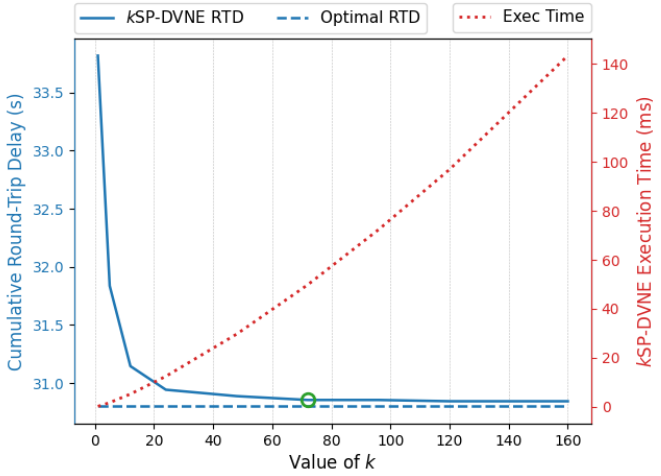
Obviously, the major advantage of the proposed heuristic is the fast computation of the distributed embedding solution while achieving almost the same round-trip delay, on average, compared with the HSAG. This fact makes the *k*SP-DVNE a suitable solution to support ultra-reliable service deployment and high availability of NFV infrastructures at the network edge, with minimized round-trip delay, especially in cases where re-configuration of the VN embedding is needed, such as end-device mobility, infrastructure failure and workload management in virtualized services. Especially, combined with the sV-VNM heuristic for the initial mapping of the VN, the *k*SP-DVNE provides most of the times the optimal solution in a real-time fashion.

6.3. Analysis for k parameter

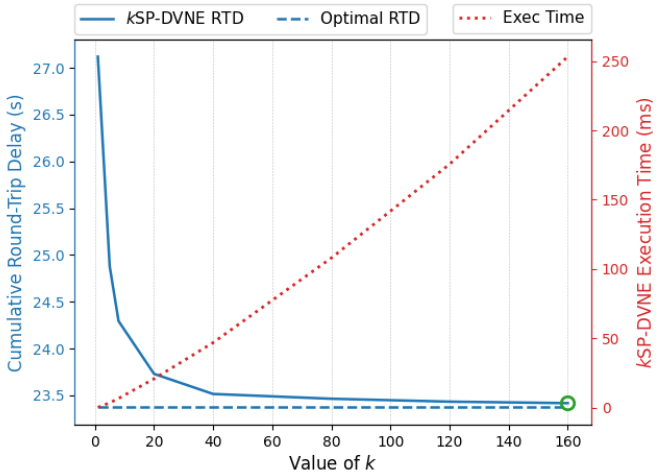
The value of the parameter k determines the number of candidate paths that will be examined by the above DVNE algorithm, while it also determines its computational complexity. Obviously, a larger value of k corresponds to a larger search space for the algorithm and increased chances for finding a near optimal, or the optimal, DVNE solution, in terms of round-trip delay. Towards finding a suitable value of parameter k, we present a simulation-based analysis. Under 4 different edge network topologies, we conduct experiments with 500 different VNE requests, with the length of the VNs varying from 5-7 following a uniform distribution, while the rest of VNE and EC simulation parameters are identical to the above *k*SP-DVNE evaluation setting. Lets us also denote by $deg(G)$ the average vertex degree of the graph G, which represents the Edge Network, and by $|V| = n$, the number of the nodes of G, representing the ECs. The characteristics of the four topologies of the simulations are as follows: The first three topologies are single-layer networks with different densities, which correspond to various numbers of EC nodes and average node degrees. More specifically, their parameters are (i) $n = 40$, $deg(G) = 3$, (ii) $n = 24$, $deg(G) = 3$ (iii) $n = 40$, $deg(G) = 4$. Also, the specifications of these topologies are the same as in the first experiment of the *k*SP-DVNE evaluation algorithm (6.2.1). The last considered topology is a hierarchical EC network, composed of three layers, following the could continuum paradigm. The layers correspond to 1) the access, 2) the aggregation, and 3) the core layers. Each of these layers has different characteristics regarding the number of ECs, the computing capacities of them, and the network delay of the links between them. We denote each layer as a subgraph G_i that contains n_i ECs. For running simulations, the parameters are set as follows: *(1) access layer*: $n_1 = 25$, $deg(G_1) = 4$, *(2) aggregation layer*: $n_2 = 10$, $deg(G_2) = 3$, and *(3) core layer*: $n_3 = 5$, $deg(G_3) = 2$. The average network link delay between the EC nodes for each layer ranges in [5, 15] ms for the G_1, [15, 30] ms for G_2, and [30, 50] ms for G_3. The network delay of the links that connect the G_1 with G_2, and G_2 with G_3 fluctuates uniformly between 20 to 40 ms, and 40 to 60 ms, respectively. Moreover, to simulate the different capacities of ECs at each layer, we assumed that the shared-VNFs that an EC is able to host are in the range [2, 3], [3, 4], and [4, 5] for the respective layers represented by G_1, G_2, G_3. Correspondingly, a different number of VNFs for every VNE request is mapped, randomly, in the EC for each layer; that is [1, 2], [2, 3], and [2, 4] VNFs of a VNE request per respective layer's EC, following a uniform distribution. In order to highlight the effect of k on the convergence and complexity of the proposed algorithm, the metrics that we consider are: (i) the cumulative round-trip delay of the *k*SP-DVNE heuristic, compared to the optimal solution, as well as (ii) its execution time, for varying values of k. Figure



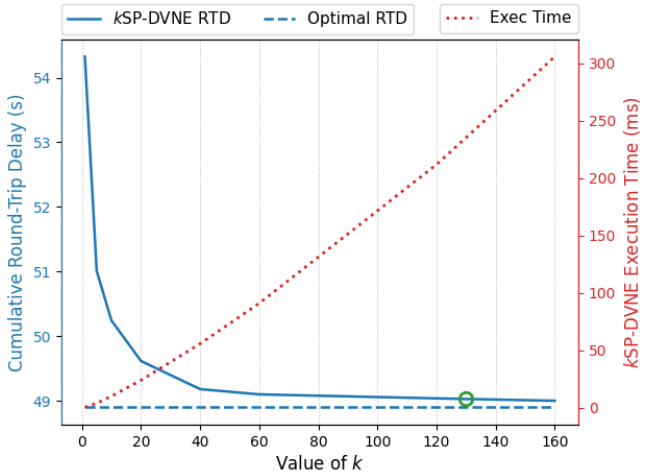
(a) 40 ECs Network with $deg(G) = 3$.



(b) 24 ECs Network with $deg(G) = 3$.



(c) 40 ECs Network with $deg(G) = 4$.



(d) 40 ECs 3-layer Hierarchical Network.

Figure 12: *k*SP-DVNE performance with varying values of k on several EC Networks.

12 shows the numerical results for the different edge network settings. At every plot, on the left y-axis the cumulative round-trip delay is shown, while the right y-axis, indicates the execution time of the *k*SP-DVNE heuristic, both with respect to the value of k. In all cases, we observe that the increase in the execution time of the algorithm is linear with respect to the value of k. The differences in the absolute execution times between the four simulations are due to the k-shortest paths algorithm's calculations, as a more dense graph with a higher number of nodes demands more time to compute the k-shortest paths. Concerning the *k*SP-DVNE convergence, as shown in Fig. 12, the algorithm achieves near optimal solutions as $k \rightarrow (n \times deg(G))$, as it pointed with circle in the corresponding sub-figures. As described earlier, the proposed algorithm will not select a DVNE solution, where multiple traversals of the same edge have to be chosen, except in the case that it is the only feasible solution in the examined path. An increased k value does not affect this behavior. Especially, in the case of the hierarchical EC network, we observe that the round-trip network delay is increased compared to the single-layer topologies, as more VNFs, shared and new deployed, have more candidate host ECs in the aggregation, or core layer, due to capacity constraints. However, based on the embedding policy of our algorithm, which avoids multiple traversals of the same link during the DVNE solution construction, its convergence to the optimal, exhibits similar behavior. Thus, in order to take advantage of the minimal execution time of the *k*SP-DVNE heuristic compared to HSAG and other exhaustive search and backtracking approaches, without discounts on its convergence, a value of k equal to $(n \times deg(G))$ is suggested. Under this setting and based on eq. (11), the computational complexity of the algorithm equals to $O(deg(G)n^4)$.

7. Conclusions

In this paper, the problem of Distributed Virtual Network Embedding in a time-efficient manner is studied. In particular, concerning the strict network delay requirements of IoT-based applications, where services are provided as VNFs to the

end-devices in an edge network, our work aims to provide efficient DVNE solutions with round-trip delay minimization. To achieve this, an algorithm that undertakes the initial Intra-EC mapping of the VN is proposed, in order to identify the candidate host ECs for each VNF of the VNE request, while at the same time ensuring to meet the computing and network resource requirements. Furthermore, an algorithm for providing the DVNE solution is introduced, aiming at round-trip delay minimization. Given the initial mapping of the VN, an augmented graph is constructed to describe the properties of the Edge Network and the VNE request, and the k shortest paths algorithm is utilized to generate candidate embedding paths for the DVNE solution construction. Considering several m pairs of f-VNP and b-VNP, km number of candidate paths are examined. The proposed algorithm's worst case complexity is $O(kn^3)$.

Regarding the initial mapping algorithm, the comparative evaluation results indicate reduced server and network link utilization within the EC infrastructure, and a higher pair VNFs co-location ratio, compared to an alternative state-of-the-art approach, especially in cases where the VN contains pre-deployed and already mapped VNFs. With reference to the kSP-DVNE, the provided comparative evaluation highlights its efficiency, as it achieves to provide the optimal solution at the 95% of the VNE requests, especially when combined with the initial Intra-EC mapping algorithm, with a significant reduction in the execution time compared to the HSAG solution. Moreover, a simulation-based analysis regarding the appropriate k value is presented.

Regarding our future work, we aim to further investigate the effect of the k parameter in the kSP-DVNE algorithm's efficiency, and the capability of the sV-VNM to approximate the optimal solution for the initial Intra-EC VN mapping. Also, we will focus on the modification of the proposed algorithms for SFCs that includes multiple paths due to load balancing functions. In addition, as the algorithm's execution time enables its usage for online re-optimization of the VNE request, and multiple VNs are able to use the same shared VNF instances, we will focus our work on extending the DVNE problem with the aspect of dynamic workload management and scaling, leveraging control theory and machine learning techniques. Finally, we will implement the proposed DVNE solution with emerging cloud resource orchestration frameworks like Kubernetes[3] and OpenStack [2] and evaluate them under realistic edge infrastructure scenarios.

Acknowledgement

This research was supported by the CHIST-ERA-18-SDCDN-003 (DRUID-NET), and has been co-funded by the European Union (European Regional Development Fund - ERDF) and Greek national funds through the Operational Program "Competitiveness, Entrepreneurship and Innovation 2014-2020" of the National Strategic Reference Framework (NSRF) - Research Funding Program: DRUID-NET (eDge computing ResoUrce allocatIon for Dynamic NETworks), MIS 5070466.

References

[1] D. Dechouniotis, N. Athanasopoulos, A. Leivadeas, N. Mitton, R. Jungers, S. Papavassiliou, Edge computing resource allocation for dynamic networks: The DRUID-NET vision and perspective, Sensors 20 (8) (2020) 2191. doi:10.3390/s20082191.

[2] OpenStack, https://www.openstack.org/, (accessed on 20 April 2023) (2022).

[3] Kubernetes, https://kubernetes.io/, (accessed on 20 April 2023) (2022).

[4] J. Halpern, C. Pignataro, Rfc 7665: Service function chaining (sfc) architecture (2015). doi:10.17487/RFC7665.

[5] ETSI GS NFV 006 V2.1.1, Architectural Framework Specification, https://www.etsi.org/deliver/etsi_gs/nfv/001_099/006/02.01.01_60/gs_nfv006v020101p.pdf, (accessed on 20 April 2023) (2021).

[6] ETSI GS MEC 003 V3.1.1, Multi-access Edge Computing (MEC); Framework and Reference Architecture, https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/03.01.01_60/gs_MEC003v030101p.pdf, (accessed on 20 April 2023) (2022).

[7] J. Dizdarević, F. Carpio, A. Jukan, X. Masip-Bruin, A survey of communication protocols for Internet of Things and related challenges of fog and cloud computing integration, ACM Computing Surveys (CSUR) 51 (6) (2019) 1–29. doi:10.1145/3292674.

[8] D. Mirkovic, G. Armitage, P. Branch, A survey of round trip time prediction systems, IEEE Communications Surveys & Tutorials 20 (3) (2018) 1758–1776. doi:10.1109/COMST.2018.2816917.

[9] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, D. Niyato, O. Dobre, H. V. Poor, 6g internet of things: A comprehensive survey, IEEE Internet of Things Journal 9 (1) (2021) 359–383. doi:10.1109/JIOT.2021.3103320.

[10] R. Mahmud, F. L. Koch, R. Buyya, Cloud-fog interoperability in iot-enabled healthcare solutions, in: Proceedings of the 19th international conference on distributed computing and networking, 2018, pp. 1–10. doi:10.1145/3154273.3154347.

[11] A. J. Ferrer, J. M. Marquès, J. Jorba, Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing, ACM Computing Surveys (CSUR) 51 (6) (2019) 1–36. doi:10.1145/3243929.

[12] D. Zheng, C. Peng, X. Liao, X. Cao, Toward optimal hybrid service function chain embedding in multiaccess edge computing, IEEE Internet of Things Journal 7 (7) (2019) 6035–6045. doi:10.1109/JIOT.2019.2957961.

[13] T. Taleb, I. Afolabi, M. Bagaa, Orchestrating 5G network slices to support industrial internet and to shape next-generation smart factories, Ieee Network 33 (4) (2019) 146–154. doi:10.1109/MNET.2018.1800129.

[14] I. Dimolitsas, D. Dechouniotis, S. Papavassiliou, P. Papadimitriou, V. Theodorou, Edge Cloud Selection: The Essential Step for Network Service Marketplaces, IEEE Communications Magazine 59 (10) (2021) 28–33. doi:10.1109/MCOM.211.2001056.

[15] B. Jamil, H. Ijaz, M. Shojafar, K. Munir, R. Buyya, Resource allocation and task scheduling in fog computing and internet of everything environments: A taxonomy, review, and future directions, ACM Computing Surveys (CSUR) 54 (11s) (2022) 1–38. doi:10.1145/3513002.

[16] G. Papathanail, A. Pentelas, I. Fotoglou, P. Papadimitriou, K. V. Katsaros, V. Theodorou, S. Soursos, D. Spatharakis, I. Dimolitsas, M. Avgeris, et al., MESON: Optimized cross-slice communication for edge computing, IEEE Communications Magazine 58 (10) (2020) 23–28. doi:10.1109/MCOM.001.2000207.

[17] Open Source MANO (OSM), "http://osm.etsi.org, (accessed on 20 April 2023) (2023).

[18] G. Papathanail, I. Dimolitsas, I. Fotoglou, D. Dechouniotis, S. Papavassiliou, P. Papadimitriou, Towards Secure and Optimized Cross-Slice Communication Establishment, in: 2022 IEEE 8th International Conference on Network Softwarization (NetSoft), IEEE, 2022, pp. 127–132. doi:10.1109/NetSoft54395.2022.9844063.

[19] K. Papadakis-Vlachopapadopoulos, I. Dimolitsas, D. Dechouniotis, E. E. Tsiropoulou, I. Roussaki, S. Papavassiliou, On blockchain-based cross-service communication and resource orchestration on edge clouds, Informatics 8 (1) (2021) 13. doi:10.3390/informatics8010013.

[20] L. Bondan, M. F. Franco, L. Marcuzzo, G. Venancio, R. L. Santos, R. J. Pfitscher, E. J. Scheid, B. Stiller, F. De Turck, E. P. Duarte, A. E. Schaeffer-Filho, C. R. P. d. Santos, L. Z. Granville, FENDE: Marketplace-Based Distribution, Execution, and Life Cycle Management of VNFs, IEEE Communications Magazine 57 (1) (2019) 13–19. doi:10.1109/MCOM.2018.1800507.

[21] P. Cappanera, F. Paganelli, F. Paradiso, VNF placement for service chaining in a distributed cloud environment with multiple stakeholders, Computer Communications 133 (2019) 24–40. doi:10.1016/j.comcom.2018.10.008.

[22] J. Pei, P. Hong, K. Xue, D. Li, Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system, IEEE Transactions on Parallel and Distributed Systems 30 (10) (2018) 2179–2192. doi:10.1109/TPDS.2018.2880992.

[23] B. Németh, N. Molner, J. Martín-Pérez, C. J. Bernardos, A. De la Oliva, B. Sonkoly, Delay and reliability-constrained vnf placement on mobile and volatile 5g infrastructure, IEEE Transactions on Mobile Computing 21 (9) (2021) 3150–3162. doi:10.1109/TMC.2021.3055426.

[24] N. Torkzaban, J. S. Baras, Trust-aware service function chain embedding: A path-based approach, in: 2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE, 2020, pp. 31–36. doi:10.1109/NFV-SDN50289.2020.9289885.

[25] D. Chemodanov, F. Esposito, P. Calyam, A. Sukhov, A constrained shortest path scheme for virtual network service management, IEEE Transactions on Network and Service Management 16 (1) (2018) 127–142. doi:10.1109/TNSM.2018.2865204.

[26] R. Chai, D. Xie, L. Luo, Q. Chen, Multi-objective optimization-based virtual network embedding algorithm for software-defined networking, IEEE Transactions on Network and Service Management 17 (1) (2019) 532–546. doi:10.1109/TNSM.2019.2953297.

[27] A. Pentelas, P. Papadimitriou, Network service embedding for cross-service communication, in: 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), IEEE, 2021, pp. 424–430, https://ieeexplore.ieee.org/abstract/document/9464038.

[28] M. Rost, S. Schmid, On the hardness and inapproximability of virtual network embeddings, IEEE/ACM Transactions on Networking 28 (2) (2020) 791–803. doi:10.1109/TNET.2020.2975646.

[29] A. Marotta, E. Zola, F. D'Andreagiovanni, A. Kassler, A fast robust optimization-based heuristic for the deployment of green virtual network functions, Journal of Network and Computer Applications 95 (2017) 42–53. doi:10.1016/j.jnca.2017.07.014.

[30] J. Y. Yen, Finding the k shortest loopless paths in a network, management Science 17 (11) (1971) 712–716. doi:10.1287/mnsc.17.11.712.

[31] R. W. Floyd, Algorithm 97: shortest path, Communications of the ACM 5 (6) (1962) 345. doi:10.1145/367766.368168.

[32] NetworkX, Network Analysis in Python, https://networkx.org (accessed on 20 April 2023) (2023).

[33] D. Laskaratos, I. Dimolitsas, G. Papathanail, M.-E. Xezonaki, A. Pentelas, V. Theodorou, D. Dechouniotis, T. Bozios, P. Papadimitriou, S. Papavassiliou, Meson: A platform for optimized cross-slice communication on edge computing infrastructures, IEEE Access 10 (2022) 49322–49336. doi:10.1109/ACCESS.2022.3171573.

[34] The University of Adelaide, The Internet Topology Zoo, http://www.topology-zoo.org/index.html (accessed on 20 April 2023) (2013).